AGNELO DENIS VIEIRA

MÉTODO DE IMPLEMENTAÇÃO DO CONTROLE DE SISTEMAS A EVENTOS DISCRETOS COM APLICAÇÃO DA TEORIA DE CONTROLE SUPERVISÓRIO

Florianópolis

2007

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

MÉTODO DE IMPLEMENTAÇÃO DO CONTROLE DE SISTEMAS A EVENTOS DISCRETOS COM APLICAÇÃO DA TEORIA DE CONTROLE SUPERVISÓRIO

Tese submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Doutor em Engenharia Elétrica.

AGNELO DENIS VIEIRA

Florianópolis, 2007

MÉTODO DE IMPLEMENTAÇÃO DO CONTROLE DE SISTEMAS A EVENTOS DISCRETOS COM APLICAÇÃO DA TEORIA DE CONTROLE SUPERVISÓRIO

Agnelo Denis Vieira

'Esta Tese foi julgada adequada para obtenção do Título de Doutor em Engenharia Elétrica, Área de Concentração em *Controle e Automação*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

_____

José Eduardo Ribeiro Cury, Dr. – UFSC
Orientador

_____

Kátia Campos de Almeida, Ph.D. – UFSC
Coordenadora do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

_____

Max Hering de Queiroz, Dr. – UFSC
Presidente

_____

Carlos Eduardo Trabuco Dórea, Dr. – UFBA

_____

André Bittencourt Leal, Dr. – UDESC

_____

Victor Juliano De Negri, Dr. – UFSC

_____

Marcelo Ricardo Stemmer, Dr. -Ing. – UFSC

À meus pais,
com carinho e gratidão.

À Alesandra e ao Luca,
fontes inestimáveis e inesgotáveis de paixão e inspiração,
sem os quais minha existência não teria sentido.

Agradecimentos

Ao professor José Eduardo Ribeiro Cury pela brilhante orientação.

Aos colegas professores do grupo de Engenharia de Produção e Mecatrônica - Produtrônica, da Pontifícia Universidade Católica do Paraná, em particular aos professores Busetti e Portela pelo empenho despendido para a viabilização do desenvolvimento desta tese.

À Pontifícia Universidade Católica do Paraná pela oportunidade de desenvolver este tese.

Resumo da Tese apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Doutor em Engenharia Elétrica.

MÉTODO DE IMPLEMENTAÇÃO DO CONTROLE DE SISTEMAS A EVENTOS DISCRETOS COM APLICAÇÃO DA TEORIA DE CONTROLE SUPERVISÓRIO

Agnelo Denis Vieira

2007

Orientador: Prof. José Eduardo Ribeiro Cury, Dr.
Área de Concentração: Controle e Automação.
Palavras-chave: sistemas a eventos discretos; teoria de controle supervisório; implementação do controle; controlador lógico programável; CLP; IEC 61131; controle distribuído.
Número de Páginas: 158

A literatura indica que, de forma geral, a prática industrial corrente para desenvolvimento de sistemas de controle em sistemas de dinâmica dirigida a eventos discretos é empírica e não adota métodos formais de modelagem do sistema nem de síntese do controlador. Esta Tese de Doutorado apresenta um método de implementação do controle de sistemas a eventos discretos. Este método permite que a implementação seja realizada concentrada em um único controlador, possivelmente um controlador lógico programável (CLP), ou distribuída em um conjunto de controladores. A lei de controle, denominada supervisor, é obtida pela aplicação da Abordagem Modular Local da Teoria de Controle Supervisório. Para aplicar esta teoria o comportamento livre do sistema e as especificações de controle são representados através de autômatos. O método de implementação é um procedimento sistemático para conversão do modelo que descreve o comportamento livre do sistema e do supervisor no programa de aplicação do CLP. Este programa de aplicação resulta em conformidade com a norma internacional IEC 61131-3. A literatura discute os diversos problemas que podem ocorrer quando os resultados da Teoria de Controle Supervisório são implementados. O método de implementação apresentado nesta Tese permite considerar tais problemas. São apresentadas propriedades que, se satisfeitas, garantem que a ocorrência de tais problemas não se manifesta. Um aspecto fundamental para a distribuição do controle é a comunicação entre os controladores. Esta Tese apresenta um modelo de comunicação entre CLPs. Este modelo garante a satisfação de um conjunto de propriedades relevantes à distribuição do controle. Este modelo emprega um serviço de comunicação definido na norma internacional IEC 61131-5.

Abstract of Thesis presented to UFSC as a partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering.

IMPLEMENTATION METHOD OF CONTROL OF DISCRETE EVENT SYSTEMS WITH APPLICATION OF THE SUPERVISORY CONTROL THEORY

Agnelo Denis Vieira

2007

Advisor: Prof. José Eduardo Ribeiro Cury, Dr.
Area of Concentration: Control and Automation.
Key-words: discrete event systems; automata; control systems; distributed control; IEC 61131
Number of Pages: 158

The literature indicates that, generally, the current industrial practice of developing control systems of discrete event dynamic systems is empirical and does not apply either formal methods of system modeling or of control synthesis. This Thesis presents a discrete event systems control implementation method. This method allows the implementation to be carried out concentrated in a single controller, possibly a programmable logic controller (PLC), or distributed in a set of controllers. The control law, named supervisor, is obtained applying the Local Modular Approach of the Supervisory Control Theory. In order to apply this theory the system free behavior and the control specifications are both represented by automata. The implementation method is a systematic procedure to convert the free behavior system model and the supervisor into the PLC application program. This application program results in conformity with the international standard IEC 61131-3. The literature discusses the several problems that may arise when the results of the Supervisory Control Theory are to be implemented. The implementation method presented in this Thesis allows to consider such problems. It is presented a set of properties that, when satisfied, guarantee that such problems do not occur. An essential aspect to the control distribution is the communication between controllers. This Thesis presents a PLC communication model that fulfils a set of properties important to control distribution. This model adopts a communication service defined by the international standard IEC 61131-5.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de símbolos

$\exists$	existe
$\forall$	para todo
$\varnothing$	conjunto vazio
|	tal que
||	operador de produto síncrono entre autômatos
|||	operador de entrelaçamento de palavras
$\alpha, \beta, \lambda, \mu, \sigma, \gamma$	eventos
s, t, u, v, w	palavras
ε	palavra de comprimento nulo
Σ	alfabeto de eventos
2^{Σ}	conjunto de todos os subconjuntos do conjunto Σ
Σ_c	alfabeto de eventos controláveis
Σ_{uc}	alfabeto de eventos não-controláveis
Σ^*	linguagem composta por todas as possíveis seqüências de eventos de comprimento finito formadas por eventos pertencentes ao alfabeto Σ
K, *M*	linguagens
$\overline{K}$	prefixo-fechamento da linguagem *K*
G = $(Q, \Sigma, \delta, q0, Q_m)$	autômato
Q	conjunto de estados do autômato **G**
Σ	alfabeto sobre o qual o autômato **G** está definido
$\delta : Q \times \Sigma \rightarrow Q$	função de transição de estados do autômato **G**
q0	estado inicial do autômato **G**
Q_m	conjunto de estados marcados do autômato **G**
$\delta(q,\sigma)!$	a função de transição de estados está definida para o par (q,σ)

$\neg\delta(q,\sigma)!$	a função de transição de estados não está definida para o par (q,σ)
$\Gamma : Q \times 2^{\Sigma}$	função eventos ativos
$L(\mathbf{G})$	linguagem gerada pelo autômato **G**
$L_m(\mathbf{G})$	linguagem marcada pelo autômato **G**
$\mathbf{G}_i = (\Sigma_i, Q_i, \delta_i, q_{0i}, Q_{mi})$ $i = \{1, ..., n\}$	representação por sistema produto, tal que $(\forall\ x,y\ \in i)$ $(\Sigma_x \cap \Sigma_y = \varnothing)$
$\mathcal{V}$: $L(\mathbf{G}) \rightarrow 2^{\Sigma}$	supervisor
$\mathfrak{I} = (S, \Phi)$	supervisor
S	autômato $(Q^S, \Sigma^S, \delta^S, q0^S, Q_m{}^S)$
$\Phi : Q^S \rightarrow 2^{\Sigma}$	mapa de saída do supervisor
$\mathcal{C}(K)$	classe de sub-linguagens de K que são **G**-controláveis
$\sup\mathcal{C}(K)$	suprema linguagem **G**-controlável de K
$\Sigma_p{}^S$	conjunto de eventos passíveis de desabilitação pelo supervisor $\mathfrak{I}$
$\Sigma_{q,q'}$	conjunto de eventos que podem causar a transição do estado q para o estado q' do autômato que representa o supervisor

1. Introdução

Conforme BRANDIN (1996), o ambiente da manufatura evoluiu de um estágio no qual era empregado um elevado grau de atividades manuais, para um estágio no qual equipamentos eram operados individualmente. Deste, para um estágio no qual os equipamentos eram capazes de realizar algumas atividades estabelecidas em seqüências automáticas, para, finalmente, um estágio com elevado grau de automação, empregando maciçamente computadores e equipamentos automáticos.

Um sistema de manufatura automatizado geralmente consiste de um determinado número de estações de processamento de materiais interconectadas por um sistema de transporte de material; um sistema de comunicação para integrar todos os aspectos da manufatura e um sistema de controle supervisório (BRANDIN, 1996). Conforme o referido autor, um sistema de controle supervisório realiza as seguintes atividades:

i) monitoramento do comportamento do sistema como um todo através de sinais de sensores transmitidos por meio de sistemas de comunicação;

ii) estabelecimento da ação de controle baseado em uma lei de controle e no estado do sistema;

iii) execução da ação de controle.

Conforme BRANDIN (1996), o desenvolvimento de um sistema de controle supervisório é constituído de três etapas principais:

i) modelagem da planta (sistema a controlar) e das especificações comportamentais a serem impostas à planta;

ii) síntese de supervisores e das leis de controle supervisório;

iii) implementação dos supervisores e das leis de controle supervisório em dispositivos de controle.

Desenvolvidos no início da década de 1970 para permitir a implementação do controle lógico de sistemas industriais, os Controladores Lógico Programáveis (CLPs) constituem-se atualmente como a principal solução tecnológica adotada no controle dos sistemas de manufatura automatizados (HELLGREN *et al.*, 2005) (FREY e LITZ, 2000).

Apesar dos esforços bem sucedidos na normatização da interface de programação dos CLPs estabelecidos pela norma internacional IEC 61131-3 (IEC, 2003), o desenvolvimento do programa de controle continua sendo realizado, no caso geral, de forma empírica com suporte na experiência adquirida pelo programador, sem adotar métodos formais de modelagem do sistema e síntese do controle. FREY e LITZ (2000) apontam que a prática industrial adotada correntemente é baseada na especificação informal do problema de controle seguida da implementação direta em uma linguagem de programação. Baseados nas informações extraídas de um questionário enviado para as principais companhias Suíças especializadas em engenharia de automação, COLLA *et al.* (2006) concluem que o setor industrial não adota qualquer metodologia ou ferramenta de suporte no desenvolvimento dos sistemas de controle industrial. Os referidos autores concluem, ainda, que a prática corrente é baseada na tomada de especificações de forma narrativa seguida da programação direta e finalmente da verificação e correção do programa obtido através de testes práticos. Contatos informais realizados pelos autores desta Tese com técnicos especializados de duas empresas líderes mundiais na fabricação de CLPs, de uma montadora multinacional de automóveis e de uma integradora de sistemas de automação com atuação globalizada confirmam este cenário no contexto nacional.

FREY e LITZ (2000) apontam as seguintes razões para a necessidade de aplicação de métodos formais na programação de CLPs, bem como na verificação e validação dos programas desenvolvidos:

i) a elevada complexidade dos problemas de controle;

ii) a crescente demanda pela redução no tempo de desenvolvimento da programação;

iii) a necessidade da possibilidade de reutilização dos programas desenvolvidos;

iv) a demanda por soluções de alta qualidade, especialmente em aplicações onde a segurança é um aspecto crítico a ser considerado.

Em junho de 2000 foi realizado na *"University of Michigan, Ann Arbor"* o *"Workshop on Logic Control for Manufacturing Systems"* o qual contou com a presença de pesquisadores renomados, ligados tanto ao setor industrial quanto à academia. O setor industrial contou com a participação de representantes de três setores: construtores de equipamentos automatizados (aqueles que desenvolvem os programas de controle), usuários finais de equipamentos automatizados e fornecedores de equipamentos para automação. Este *workshop* teve como objetivo estabelecer um entendimento sobre a lacuna existente entre a teoria de sistemas a eventos discretos e sua aplicação no controle de sistemas de manufatura industriais.

Do relatório das atividades realizadas neste *workshop* (TILBURY e KHARGONEKAR, 2000) extrairam-se os trechos mais relevantes que caracterizam a situação atual e reforçam os

argumentos apresentados nos parágrafos anteriores. Este relatório apresenta ainda o encaminhamento futuro no tocante ao projeto e implementação de sistemas de controle.

> "Tipicamente os sistemas de manufatura consistem de diversos equipamentos operando de forma seqüencial e coordenada. Controladores Lógico Programáveis (CLPs) são amplamente utilizados para implementar os algoritmos de controle destes equipamentos. Sistemas com centenas ou milhares de entradas e saídas não são incomuns. O controle lógico deve considerar não apenas as situações de operação normal, mas também prever interfaces com o operador e considerar situações de erro e recuperação.
>
> Na prática industrial corrente, os CLPs são programados utilizando linguagens de baixo nível, tipicamente diagramas escada, o que resulta em programas muito extensos e de difícil compreensão e manipulação. O projeto do sistema de controle como um todo é realizado com base na experiência do projetista, e a verificação do mesmo é tipicamente realizada apenas através de experimentação ou simulação. Devido à complexidade dos programas de controle e dos sistemas de manufatura, tanto a verificação por experimentação quanto por simulação demanda muito tempo e recursos financeiros.
>
> Fundamentos teóricos para modelagem e controle de sistemas cuja dinâmica é determinada pela ocorrência de eventos discretos têm sido desenvolvidos. Os dois formalismos mais comuns são Redes de Petri e Máquinas de Estados Finitos; ambos permitem a verificação formal da correção do sistema de controle. Entretanto, apesar dos avanços significativos ocorridos nos últimos anos, métodos formais não têm sido empregados de forma significativa na indústria.
>
> ...
>
> Tópicos para pesquisas futuras incluem o aprimoramento do projeto e implementação de programas de controle lógico através do diagnóstico integrado, melhores interfaces homem-máquina, validação e geração automática de código. Na medida em que a implementação distribuída do controle se torne mais popular, serão necessárias técnicas para projeto e análise de sistemas de controle lógico distribuído. Atualmente não existem métodos efetivos para composição de sistemas de controle distribuído, geralmente a distribuição do controle aumenta a complexidade da estrutura de controle."

SARDESAI *et al.* (2006) destacam que o controle dos sistemas de manufatura tradicionais é tipicamente centralizado em um CLP. Destacam, também, a crescente necessidade de sistemas de manufatura ágeis e flexíveis. De acordo com estes autores, para satisfazer estas características os sistemas de manufatura devem ser capazes de se adaptar e reconfigurar em função das modificações do ambiente, devem ser capazes de ser reutilizados em conjunto com outros sistemas de manufatura, devem reagir em tempo real enquanto preservam a segurança e a confiabilidade. Para tanto é necessário uma mudança de paradigma, evoluindo de uma estrutura de controle centralizada em um CLP para uma arquitetura de controle distribuído. Cada componente no sistema

deve possuir seu próprio controlador, o qual deve ser capaz de se comunicar com os outros dispositivos.

Os argumentos apresentados no parágrafo anterior são reforçados pelo exposto em (CENGIC *et al*., 2005). De acordo com os autores deste trabalho, nos sistemas desenvolvidos pelo homem há a necessidade de sistemas de controle distribuído. A necessidade é maior em casos em que o sistema a ser controlado é geograficamente distribuído ou disperso sobre uma grande área. Exemplos de tais sistemas são os sistemas de manufatura e os processos químicos. Porém, conforme CENGIC *et al.* (2005), o desenvolvimento de um sistema de controle distribuído é custoso e sujeito a erros.

Conforme VYATKIN *et al.* (2006) os seguintes aspectos justificam a razão pela qual a automação distribuída se tornou um foco de pesquisa na última década:

i) Performance. Quando a performance de um controlador não é suficiente, em geral, é uma boa idéia dividir o programa de controle e implementá-lo em diferentes controladores. Em geral, o tempo de resposta de um CLP é linearmente dependente em relação ao tamanho do programa, portanto, tal divisão deve resultar em um ganho imediato na produtividade. Tal decomposição requer uma definição explícita de um protocolo de transmissão de mensagens ou o uso de uma base de dados comum;

ii) Distribuição espacial. Para os sistemas que são espacialmente distribuídos, a leitura do estado dos sensores e o envio de comandos para os atuadores são realizados através de redes de comunicação com protocolos industriais. Isto pode resultar em atrasos consideráveis entre a detecção de eventos e o estabelecimento de ações de controle. A solução óbvia é delegar a tomada de decisão para um controlador localizado próximo ao equipamento a ser controlado;

iii) Facilidade de integração e reutilização. Em muitos casos os sistemas de manufatura são constituídos de sub-sistemas mecatrônicos com algum sistema de controle incorporado. Tais controladores devem ser preservados e integrados ao sistema global.

VYATKIN *et al.* (2006) concluem que há duas abordagens que conduzem a sistemas de controle distribuído. A primeira abordagem consiste na decomposição de uma estrutura de controle centralizada em diversos controladores distribuídos que se comunicam entre si. A segunda abordagem consiste na integração de controladores dedicados a determinados subsistemas ao controle do sistema com um todo.

Esta Tese de Doutorado define um método de implementação do controle de sistemas a eventos discretos. Entende-se que, por si só, a definição e divulgação de métodos formais não resultam na alteração da prática industrial com relação à implementação do controle de sistemas a

eventos discretos, porém, são condições necessárias para tanto. Nos próximos parágrafos é realizada uma breve caracterização do método proposto.

Como método formal para modelagem do sistema a ser controlado e síntese dos supervisores é adotada uma extensão da Teoria de Controle Supervisório (RAMADGE e WONHAM, 1987). Tal extensão é denominada Abordagem Modular Local e foi proposta por QUEIROZ e CURY (2000a, 2000b, 2002a).

A Teoria de Controle Supervisório é uma importante ferramenta para o controle dos sistemas a eventos discretos. Baseado num modelo que descreve o comportamento livre do sistema a ser controlado e num conjunto de especificações comportamentais é possível realizar a síntese formal de um supervisor. A ação de controle deste supervisor restringe o comportamento do sistema de forma a satisfazer o conjunto de especificações. De acordo com a Abordagem Modular Local, ao invés de realizar a síntese de um único supervisor que atende ao conjunto de especificações como um todo, é realizada a síntese de um conjunto de supervisores modulares, cada qual satisfazendo uma especificação em separado, tal que a ação conjunta de todos os supervisores modulares satisfaça ao conjunto de especificações como um todo. A Abordagem Modular Local considera e preserva a modularidade natural do sistema a ser controlado e das especificações de controle na síntese do conjunto de supervisores.

Conforme HELLGREN *et al.* (2005), apesar da grande aceitação da Teoria de Controle Supervisório pelo meio acadêmico, havendo diversas extensões à mesma e um número muito grande de publicações com foco em aspectos teóricos, são raras as aplicações industriais. A razão principal para isto é o problema da implementação física. Há poucas referências de como implementar os supervisores obtidos através da aplicação desta teoria. No caso da implementação em CLPs "*the gap between the event-based automata world and the signal-based PLC-world has to be bridged*".

QUEIROZ e CURY (2002b) definem uma Arquitetura de Controle Supervisório estruturada em três níveis. Esta arquitetura impõe a ação de controle estabelecida pelo conjunto de supervisores obtidos com a aplicação da Teoria de Controle Supervisório e atua como uma interface entre o modelo teórico empregado na síntese dos supervisores e o sistema a ser controlado.

O método de implementação definido nesta Tese estabelece como implementar a Arquitetura de Controle Supervisório mencionada no parágrafo anterior. A implementação pode ser realizada de forma concentrada em um único CLP ou distribuída em um conjunto de CLPs. Este método permite ao projetista converter sistematicamente o conjunto de supervisores e o conjunto de autômatos que descrevem o comportamento livre do sistema a ser controlado em um conjunto

de *Sequential Function Charts*. O método também enumera e detalha os diversos componentes do programa de aplicação a ser implementado em cada CLP. O programa obtido com a aplicação deste método resulta em conformidade com a norma internacional IEC 61131-3 (IEC, 2003). Este método permite que sejam desenvolvidos aplicativos para geração automática de código, o que resulta na redução do tempo necessário para desenvolvimento da solução de controle bem como minimiza, ou mesmo elimina, a possibilidade de erros durante a etapa de geração de código.

A adoção deste método de implementação resulta em:

i) facilidade de interpretação do código;

ii) facilidade de avaliação e correção do código;

iii) facilidade de alteração do código em função de:

- inclusão ou exclusão de subsistemas;

- alteração das especificações de controle;

- alteração da solução tecnológica adotada para realizar determinadas atividades e tarefas;

iv) facilidade de reaproveitamento de código para outras implementações;

v) uso racional de memória do CLP.

Este documento está organizado da seguinte forma: o Capítulo 2 apresenta alguns conceitos fundamentais referentes à caracterização e representação do comportamento de sistemas, introduz a classe de sistemas foco de estudo ao longo deste trabalho, a classe dos "Sistemas a Eventos Discretos", e finalmente apresenta os conceitos fundamentais sobre Linguagens Formais e Autômatos empregados ao longo deste trabalho para representação e análise do comportamento dos sistemas pertencentes à referida classe. No Capítulo 3 são apresentados conceitos relativos à Teoria de Controle Supervisório, tanto na abordagem centralizada quanto na abordagem Modular Local. O Capítulo 4 apresenta aspectos conceituais, tecnológicos e normativos relativos à implementação do controle de sistemas a eventos discretos. Ao final deste capítulo é apresentada uma arquitetura de referência para realizar a implementação do controle de sistemas a eventos discretos com base na aplicação da Teoria de Controle Supervisório. No Capítulo 5 é apresentado o método de implementação da referida arquitetura de controle, com foco na implementação concentrada em um único CLP. No Capítulo 6 é proposto um modelo de comunicação entre CLPs. No Capítulo 7 o método definido no Capítulo 5 é estendido de forma a permitir a implementação distribuída em um conjunto de CLPs. Finalmente, no Capítulo 8 são apresentadas considerações finais e encaminhamentos de trabalhos futuros.

2. Sistemas a eventos discretos

Apresentam-se na seção inicial deste capítulo conceitos fundamentais referentes à caracterização e representação de sistemas. Nesta seção é introduzida a classe de sistemas foco de estudo deste trabalho, denominada "Sistemas a Eventos Discretos". Nas seções seguintes são introduzidos os principais formalismos adotados neste trabalho para representação e análise do comportamento dos sistemas pertencentes à referida classe.

2.1 Conceituação geral

O termo sistema apresenta diversas definições e pode ser aplicado a diferentes áreas do conhecimento. De forma geral, este termo descreve o conceito de agregação de diversos componentes com o objetivo de realizar determinadas funções que não poderiam ser executadas pelos componentes isolados. Conforme definido em (FERREIRA, 1999), sistema é a "disposição das partes ou dos elementos de um todo, coordenados entre si, e que funcionam como estrutura organizada". De acordo com *IEEE - Standard Dictionary of Electrical and Electronic Terms apud* (CASSANDRAS e LAFORTUNE, 1999) sistema é "*a combination of components that act together to perform a function not possible with any of the individual parts*".

Um conceito fundamental no estudo dos sistemas é o descrito pelo termo modelo. Conforme WILSON (1990) *apud* DE NEGRI (1996) modelo é "a interpretação explícita do entendimento de uma situação, ou meramente das idéias acerca daquela situação. Este pode ser expresso matematicamente, por símbolos ou palavras, mas essencialmente é uma descrição de entidades, processos ou atributos e as relações entre eles." ATTIÉ (1998) conclui, "qualquer modelo é, em essência, parcial e fruto de determinada abstração, na medida em que apresenta sempre o sistema sob determinado enfoque, privilegiando um ou outro aspecto deste, enfatizando certos detalhes ou propriedades enquanto outros são suprimidos."

No contexto deste trabalho, o estudo dos sistemas tem por objetivo a representação dos mesmos através de modelos formais que permitam descrever de forma satisfatória seu comportamento. Uma vez que este comportamento pode violar determinadas especificações comportamentais, deseja-se sintetizar e implementar leis de controle que atuem sobre o sistema de

forma que seu comportamento sob a ação de controle seja o mais próximo possível do comportamento desejado, ou seja, daquele comportamento que satisfaz as referidas especificações. Pelo emprego de fundamentos teóricos e ferramentas computacionais busca-se, ainda, analisar o comportamento do sistema de forma a verificar a satisfação de determinadas propriedades. Destaca-se que a realização destes objetivos não depende unicamente do domínio de fundamentos teóricos e ferramentas computacionais, pois a base do processo está fundamentada na interpretação e compreensão do sistema associado à capacidade de abstração de detalhes irrelevantes e à correta representação dos aspectos realmente relevantes.

Em (CASSANDRAS e LAFORTUNE, 1999) é realizada a classificação dos sistemas em diversas categorias. Os próprios autores tornam explícito que esta classificação não é excludente, pois depende basicamente da perspectiva empregada para interpretar e compreender o sistema.

Os sistemas denominados “sistemas dinâmicos a variáveis contínuas” ou simplesmente “sistemas contínuos” caracterizam-se basicamente por dois fatores:

i) o espaço de estados é contínuo, isto é, as variáveis do sistema podem assumir qualquer valor dentro de um determinado intervalo de variação contínuo;

ii) o comportamento das variáveis do sistema é regido pelo tempo.

De forma geral, o formalismo matemático para estudo dos sistemas contínuos se baseia em equações diferencias ou em equações a diferença. Neste ponto o leitor é incentivado a realizar a leitura do Capítulo 1 de (CASSANDRAS e LAFORTUNE, 1999) onde são definidos e exemplificados de forma sucinta conceitos fundamentais à teoria de sistemas.

Em contraposição aos “sistemas contínuos” os sistemas denominados “sistemas dinâmicos a eventos discretos” ou da forma mais usual “sistemas a eventos discretos” (SEDs) apresentam as seguintes características:

i) o espaço de estados é discreto, ou seja, as variáveis do sistema podem assumir valores pré-estabelecidos pertencentes a um conjunto discreto;

ii) o comportamento das variáveis independe do tempo e é dirigido por eventos.

O termo “evento” é empregado para descrever a ocorrência, abrupta e sem duração no tempo, de um fenômeno no sistema em estudo ou no ambiente no qual o mesmo está inserido e que pode afetar o comportamento deste sistema. A cada ocorrência de um evento o sistema pode assumir um novo comportamento ou executar uma nova função, ou seja, o sistema pode assumir um novo estado. Assume-se que a cada instante de tempo só pode ocorrer um único evento.

De forma geral, para um SED são válidas as seguintes observações:

– a ocorrência de eventos é assíncrona no tempo;

– o estado do sistema permanece imutável até que ocorra um evento;

– para um dado estado do sistema, a ocorrência de um determinado evento não implica necessariamente a mudança de estado;

Um sistema determinístico quando submetido, a partir de um determinado estado, a uma mesma seqüência de eventos apresenta sempre a mesma trajetória no espaço de estados, isto é, a seqüência de estados visitados é sempre a mesma.

Se for conhecido o estado inicial de um sistema determinístico, seu comportamento pode ser representado através de uma seqüência de eventos. O diagrama apresentado na Figura 2.1 descreve o comportamento de um sistema que pode assumir dois estados: REPOUSO e OPERAÇÃO. A ocorrência do evento α conduz o sistema para o estado OPERAÇÃO, e a ocorrência do evento β conduz o sistema para o estado REPOUSO. Nesta representação, o sistema está inicialmente no estado REPOUSO e ocorre a seqüência de eventos $\alpha\beta\alpha\alpha\beta$.

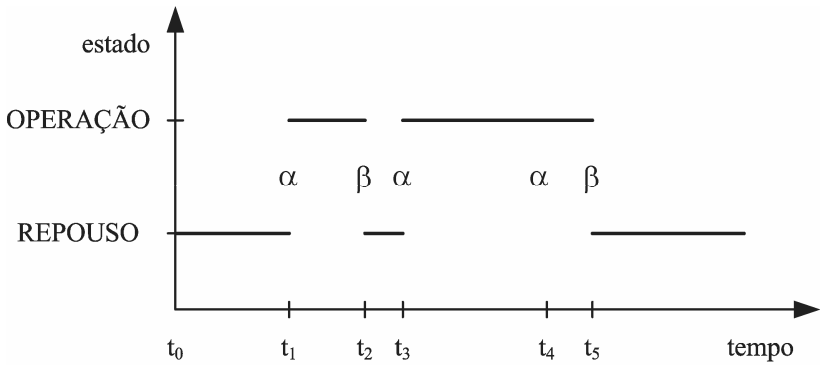


Figura 2.1 – Trajetória de um sistema no espaço de estados

O comportamento do sistema descrito no parágrafo anterior é uma representação não temporizada do mesmo. Esta representação é empregada quando o objetivo é considerar aspectos qualitativos do sistema, determinados pela ordem em que os eventos ocorrem, ou seja, o comportamento lógico do mesmo. Um exemplo de grande interesse no estudo do comportamento lógico é a coordenação da operação conjunta de equipamentos industriais de forma a garantir que uma determinada ordem de ocorrência de eventos seja satisfeita. Através da análise do comportamento lógico de um sistema é possível verificar se um determinado estado é alcançado.

Quando o objetivo é considerar quantitativamente o tempo em que os eventos ocorrem, devem ser empregados modelos temporizados. A descrição temporizada pode ser realizada através da seqüência de pares ordenados (evento, instante em que ocorre o evento). O comportamento representado na Figura 2.1 poderia ser descrito da seguinte forma: estando o sistema no estado

REPOUSO a ocorrência da seqüência temporizada de eventos (α, t1) (β, t2) (α, t3) (α, t4) (β, t5) conduz o sistema ao estado REPOUSO a partir do instante t5. Através de um modelo temporizado é possível analisar propriedades referentes ao comportamento dinâmico em tempo real.

A descrição do comportamento do sistema pode ser ainda mais refinada se forem incluídas à seqüência de eventos temporizada, informações acerca da probabilidade de ocorrência de eventos futuros. Uma descrição deste tipo representa o comportamento temporizado estocástico do sistema. Modelos que descrevem este comportamento são úteis para questões relacionadas à análise de desempenho.

No contexto dos sistemas a eventos discretos não existe um formalismo matemático único que seja satisfatório para a consideração simultânea de aspectos presentes em todas as classes de problemas que se podem formular para os SEDs. A obra "*Introduction to Discrete Event System"* (CASSANDRAS e LAFORTUNE, 1999) apresenta uma introdução aos principais formalismos matemáticos atualmente empregados no estudo de SEDs, dentre as quais citam-se: Linguagens formais; Autômatos; Controle supervisório; Redes de Petri; Álgebra de Dióides; Cadeias de Markov e Teoria das filas.

Uma abordagem recente no estudo dos sistemas permite considerar simultaneamente as características contínuas e discretas de seus componentes e de suas inter-relações. Segundo esta abordagem os sistemas podem ser classificados como "Sistemas Híbridos".

2.2 Linguagens Formais e Autômatos

Uma possível forma de representar o comportamento de um SED é através de uma tabela de transição de estados. Tal tabela informa qual estado é alcançado quando o sistema está em um determinado estado e ocorre um certo evento. Contudo, este não é o procedimento usualmente empregado, pois não confere ao projetista procedimentos formais para composição de modelos de sistemas formados pela interação de múltiplos subsistemas; para análise do comportamento do sistema; para síntese de controladores, dentre outros. Visto que, conhecido o estado inicial, o comportamento lógico de um SED pode ser descrito através da seqüência de eventos gerada, e se considerarmos que eventos representam elementos de um alfabeto e que seqüências de eventos representam palavras sobre este alfabeto, pode-se descrever o comportamento do sistema através de uma determinada linguagem.

Conforme apresentado em (CASSANDRAS e LAFORTUNE, 1999) "... a abordagem empregando a teoria de linguagens é atrativa para apresentar aspectos da modelagem e para discutir propriedades de SEDs. Entretanto, ela não é conveniente para realizar a verificação de propriedades ou a síntese do controlador. O que também é necessário é uma forma conveniente de representá-la.

Se a linguagem for finita é sempre possível listar todos os seus elementos, ou seja, todas as possíveis seqüências de eventos. Infelizmente isto raramente é factível. Preferencialmente, gostaríamos de empregar um formalismo que permitisse representar a linguagem de uma forma que destacasse a estrutura do comportamento do sistema e que fosse conveniente de manipular quando se deseja realizar a verificação de propriedades e a síntese do controlador." Dois formalismos são apresentados na referida obra: Autômatos e Redes de Petri. Estes formalismos apresentam em comum a capacidade de representar linguagens através de estruturas de transição de estados. Além disto, possuem operações formais que permitem que o modelo utilizado para representar cada componente do sistema seja utilizado para compor o modelo do sistema como um todo. Permitem, ainda, a análise do comportamento, a verificação de propriedades e a síntese do controlador.

Como base teórica para compreensão da Teoria de Controle Supervisório (RAMADGE e WONHAM, 1987) destacam-se as teorias sobre Linguagens Formais, Expressões Regulares e Autômatos. Tendo em vista a extensão destes assuntos e a disponibilidade de referências completas sobre os mesmos, nos restringimos a apresentar apenas seus conceitos fundamentais. O leitor interessado em obter maiores informações pode consultar as seguintes referências (CARROLL e LONG, 1989), (CASSANDRAS e LAFORTUNE, 1999), (CURY, 2001), (HOPCROFT *et al.*, 2001), (KUMAR e GARG, 1995), (QUEIROZ, 2000) e (WONHAM, 2003).

A Teoria de Controle Supervisório representa o comportamento livre de um SED através de um autômato na forma $(Q, \Sigma, \delta, q0, Q_m)$, onde:

Q – representa o conjunto de estados utilizados para descrever o comportamento do sistema segundo a abstração empregada;

Σ – representa o conjunto de eventos relevantes, usualmente denominado alfabeto de eventos;

q0 – representa o estado inicial do sistema, sendo que $q0 \in Q$;

$\delta : Q \times \Sigma \rightarrow Q$ – representa a função de transição de estados, sendo que esta função pode ser definida para apenas alguns pares ordenados em $Q \times \Sigma$;

Q_m – representa um conjunto de estados marcados, sendo que $Q_m \subseteq Q$;

Emprega-se a notação $\delta(q,\sigma)!$ para especificar que a função é definida para o par ordenado (q, σ). A notação $\neg\delta(q,\sigma)!$ é utilizada em caso contrário.

Há casos em que um subconjunto de estados do sistema apresenta uma semântica diferenciada dos demais estados. É possível diferenciar este subconjunto através da sua marcação. Isto é realizado especificando um subconjunto não-vazio $Q_m \subseteq Q$. No âmbito do controle supervisório utiliza-se a marcação de estados para representar a conclusão de tarefas ou especificar objetivos de controle, ou seja, aquilo que o sistema deve ser capaz de realizar sob ação de controle.

Quando o número de estados do autômato é finito, é possível realizar sua representação através de um diagrama de transição de estados, o qual é um grafo direcionado onde os nós representam os estados do autômato. A cada tripla ordenada da função de transição de estados, na forma (q, σ, q'), é associado um arco direcionado que leva do estado de origem (q) ao estado de destino (q'). Cada arco é identificado com o evento (σ) que provoca a correspondente transição de estado. A representação de um nó empregando uma linha dupla é usualmente utilizada para identificar que o estado pertence ao subconjunto de estados marcados. Os estados não-marcados são usualmente representados com linha simples. O estado inicial é identificado através de um arco direcionado que conduz ao nó correspondente, mas cuja origem não está interligada a qualquer outro nó. A este arco não é associada qualquer identificação.

A um autômato pode ser associada uma função eventos ativos $\Gamma : Q \rightarrow 2^{\Sigma}$, a qual indica o conjunto de eventos que podem ocorrer a partir de um determinado estado, ou seja, o conjunto de eventos no estado $q \in Q$ tal que $\delta(q, \sigma)!$.

Sendo Σ o alfabeto de eventos relevantes para descrever o comportamento do sistema, uma seqüência destes eventos corresponde a uma palavra sobre este alfabeto. O comprimento da palavra *s* corresponde ao número de ocorrências de eventos na referida seqüência e é designada por |*s*|. Uma palavra de comprimento nulo, ou seja, uma seqüência de eventos vazia é usualmente designada por ε. O conjunto Σ^* representa a linguagem composta por todas as possíveis seqüências de eventos de comprimento finito formadas por eventos pertencentes ao alfabeto Σ e a palavra de comprimento nulo. Todo subconjunto de Σ^* constitui uma linguagem sobre Σ, em particular a linguagem vazia (representada por ∅), ε e Σ^* são linguagens em Σ.

Por conveniência, a função de transição de estados é estendida do domínio Q x Σ para o domínio Q x Σ^*, o que permite processar seqüências de eventos ao invés de processar individualmente cada evento. A função $\hat{\delta}: Q \times \Sigma^* \rightarrow Q$ é definida como:

$\hat{\delta}(q, \varepsilon) = q$

$(\forall \sigma \in \Sigma)\, \hat{\delta}(q, \sigma) = \delta(q, \sigma)$

$(\forall \sigma \in \Sigma, s \in \Sigma^*)\, \hat{\delta}(q, s\sigma) = \delta(\hat{\delta}(q, s), \sigma)$

Seja **G** um autômato definido sobre o alfabeto Σ. A este autômato são associadas duas linguagens, denominadas linguagem gerada e linguagem marcada e representadas, respectivamente, como $L(\mathbf{G})$ e $L_m(\mathbf{G})$, tal que $L_m(\mathbf{G}) \subseteq L(\mathbf{G}) \subseteq \Sigma^*$. A linguagem gerada representa o conjunto de todas as possíveis seqüências de eventos que podem ser geradas pelo sistema representado pelo autômato em questão. A linguagem marcada representa as seqüências de eventos que conduzem à realização de uma tarefa, ou seja, todas as seqüências de eventos que, partindo do estado inicial,

conduzem a um estado marcado. Caso a função de transição de estados esteja definida para todos os pares ordenados em Q x Σ, então $L(\mathbf{G}) = \Sigma^*$. Tais linguagens são definidas como:

$$L(\mathbf{G}) = \{s \mid s \in \Sigma^* \text{ e } \delta(q_0, s)!\}$$
$$L_m(\mathbf{G}) = \{s \mid s \in \Sigma^* \text{ e } \delta(q_0, s) \in Q_m\}$$

Sejam **G1** e **G2** dois autômatos quaisquer. Tais autômatos são equivalentes se $L(\mathbf{G1}) = L(\mathbf{G2})$ e $L_m(\mathbf{G1}) = L_m(\mathbf{G2})$ (CASSANDRAS e LAFORTUNE, 1999).

2.3 Operações sobre linguagens e autômatos

Sejam *s* e *t* duas palavras definidas sobre Σ^*, a concatenação destas palavras é a palavra formada pela seqüência de eventos que compõem a palavra *s* seguida da seqüência de eventos que compõem a palavra *t*. A concatenação destas palavras pode ser representada como *st*.

Toda seqüência de eventos que constitui a parte inicial de uma palavra é denominada prefixo desta palavra. Em particular, a palavra de comprimento nulo é um prefixo de toda palavra, além disto, toda palavra é um prefixo dela mesma.

O prefixo-fechamento de uma linguagem é definido como o conjunto formado por todos os prefixos das palavras pertencentes à linguagem em questão. O prefixo-fechamento de uma linguagem *K* é denotado por $\overline{K}$. De forma geral $K \subseteq \overline{K}$, ou seja, uma linguagem é igual ou está contida no seu prefixo-fechamento.

Sendo **G** um autômato definido sobre o alfabeto Σ, têm-se a seguinte relação entre linguagens $L_m(\mathbf{G}) \subseteq \overline{L_m(\mathbf{G})} \subseteq L(\mathbf{G}) = \overline{L(\mathbf{G})} \subseteq \Sigma^*$.

Um estado de um autômato é denominado acessível se existe uma seqüência de eventos que conduz do estado inicial deste autômato ao estado em questão. Um estado é denominado coacessível se existe uma seqüência de eventos que conduz deste estado a um estado marcado. Se todos os estados forem acessíveis, o autômato é acessível. Se todos os estados forem coacessíveis, o autômato é coacessível. O autômato é TRIM se for simultaneamente acessível e coacessível. É possível obter a componente acessível de um autômato através da eliminação dos estados não-acessíveis e das transições associadas a eles. De forma similar, é possível obter a componente coacessível de um autômato através da eliminação dos estados não-coacessíveis e das transições associadas. Sendo **G** um autômato, Ac(**G**) denota a operação que obtém a componente acessível do referido autômato.

Se a componente acessível do autômato for coacessível então está assegurada a igualdade $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$ e o autômato é não-bloqueante. Em caso contrário, a inclusão $\overline{L_m(\mathbf{G})} \subset L(\mathbf{G})$ é

própria e existe pelo menos um estado acessível a partir do qual nunca é possível concluir uma tarefa (atingir um estado marcado). Neste caso o autômato é bloqueante.

A intersecção de duas linguagens resulta na linguagem formada por palavras que pertencem, simultaneamente, às duas linguagens originais. A intersecção das linguagens K_A e K_B é denotada por $K_A \cap K_B$.

Sejam $\mathbf{G_1} = (Q^{G1}, \Sigma^{G1}, \delta^{G1}, q0^{G1}, Q_m{}^{G1})$ e $\mathbf{G_2} = (Q^{G2}, \Sigma^{G2}, \delta^{G2}, q0^{G2}, Q_m{}^{G2})$ dois autômatos quaisquer. O autômato **G** obtido pelo produto síncrono destes autômatos é definido por:

$$\mathbf{G} = \mathbf{G_1} \parallel \mathbf{G_2} = Ac((Q^{G1} \times Q^{G2}) , (\Sigma^{G1} \cup \Sigma^{G2}) , \delta^{G} , (q0^{G1}, q0^{G2}) , (Q_m{}^{G1}, Q_m{}^{G2}))$$

A função de transição de estados $\delta^G : (Q^{G1} \times Q^{G2}) \times (\Sigma^{G1} \cup \Sigma^{G2}) \rightarrow (Q^{G1} \times Q^{G2})$ é definida como:

$$\delta^G((q^{G1},q^{G2}),\sigma) := \begin{cases} (\delta^{G1}(q^{G1},\sigma) , \delta^{G2}(q^{G2},\sigma)) & \text{se : } \sigma \in (\Sigma^{G1} \cap \Sigma^{G2}) \text{ e } \delta^{G1}(q^{G1},\sigma)! \text{ e } \delta^{G2}(q^{G2},\sigma)! \\ (\delta^{G1}(q^{G1},\sigma) , \quad q^{G2} \quad) & \text{se : } \sigma \in \Sigma^{G1} \text{ e } \sigma \notin \Sigma^{G2} \text{ e } \delta^{G1}(q^{G1},\sigma)! \\ (\quad q^{G1} \quad , \delta^{G2}(q^{G2},\sigma)) & \text{se : } \sigma \notin \Sigma^{G1} \text{ e } \sigma \in \Sigma^{G2} \text{ e } \delta^{G2}(q^{G2},\sigma)! \\ \text{indefinida} & \text{em caso contrário} \end{cases}$$

A definição da função de transição de estados estabelece que os eventos comuns aos dois autômatos só podem ocorrer se puderem ser executados simultaneamente pelos dois autômatos. Os eventos exclusivos a um determinado autômato podem ser executados sempre que possível. Assim, os autômatos $\mathbf{G_1}$ e $\mathbf{G_2}$ estão sincronizados pelos eventos comuns.

Caso o alfabeto dos dois autômatos seja idêntico, então $L(\mathbf{G}) = L(\mathbf{G_1}) \cap L(\mathbf{G_2})$ e $L_m(\mathbf{G}) = L_m(\mathbf{G}_1) \cap L_m(\mathbf{G_2})$. Caso os autômatos não possuam eventos em comum, então o autômato **G** representa o comportamento concorrente e assíncrono dos dois autômatos.

Baseados no exposto em HOARE (1985) FABIAN e HELLGREN (1998) definem o entrelaçamento (*interleaving*) de duas seqüências de eventos $s1, s2 \in \Sigma^*$ como a seqüência obtida pela concatenação de um prefixo de uma das seqüências com um prefixo da outra seqüência concatenado com o entrelaçamento dos sufixos remanescentes destas seqüências. O conjunto de seqüências denotado por $s1 \parallel\parallel s2$ é a linguagem obtida pelos possíveis entrelaçamentos de *s1* e *s2*. Sejam *s1* = *abc* e *s2* = *xy* duas seqüências de eventos, então $s1 \parallel\parallel s2$ = { *abcxy*, *abxyc*, *abxcy*, *axybc*, *axbyc*, *axbcy*, *xyabc*, *xaybc*, *xabyc*, *xabcy* }.

2.4 Sistemas Compostos

WILLNER e HEYMANN (1991) afirmam que a maior parte dos sistemas a eventos discretos reais é concorrente, isto é, é constituído de um grande número de subsistemas que operam

concorrentemente. Tais sistemas também são designados sistemas compostos. Quanto maior o detalhamento necessário para realizar a descrição do comportamento de tais sistemas, maior a complexidade para construir um modelo único que descreva de forma adequada e fiel este comportamento. Uma abordagem natural para representar o comportamento de sistemas compostos consiste em explorar a modularidade e distribuição naturais do mesmo, representando o comportamento independente de cada subsistema através de um autômato.

Conforme WILLNER e HEYMANN (1991), caso os conjuntos de eventos dos autômatos utilizados nesta representação sejam disjuntos, então, os sistemas operam concorrentemente, mas assincronamente. Caso o comportamento de dois ou mais subsistemas seja descrito através de eventos comuns, haverá uma sincronização através destes eventos, ou seja, a ocorrência de um evento é simultânea em todos os subsistemas que compartilham o referido evento. Este comportamento é capturado pela definição de produto síncrono de autômatos.

Se a representação de cada subsistema é realizada através de um autômato na forma $\mathbf{G_i} = (Q^{Gi}, \Sigma^{Gi}, \delta^{Gi}, q0^{Gi}, Q_m{}^{Gi})$, com $i \in I = \{1, ..., n\}$ onde n corresponde ao número de subsistemas, o comportamento do sistema pode ser representado pelo autômato $\mathbf{G} = (Q^G, \Sigma^G, \delta^G, q0^G, Q_m{}^G)$ obtido pelo produto síncrono de todos autômatos $\mathbf{G_i}$, denotado da seguinte forma $\mathbf{G} = \|_{i=1}^{n} \mathbf{G_i}$. O modelo assim obtido é denominado Representação por Sistema Composto. Caso estes autômatos não apresentem eventos em comum, o modelo é denominado Representação por Sistema Produto (RAMADGE e WONHAM, 1989).

2.5 Conclusão

Caracterizou-se neste capítulo a classe de sistemas foco de estudo ao longo desta Tese, classe esta denominada Sistemas a Eventos Discretos. Apresentou-se, também, os formalismos que serão empregados na representação do comportamento livre destes sistemas. Em particular, verifica-se que a representação do comportamento de sistemas constituídos pela operação concorrente de diversos subsistemas é realizada de forma conveniente através de representações por sistema composto e representações por sistema produto. Tais representações simplificam a descrição do comportamento livre do sistema, pois são obtidas através da composição dos modelos dos subsistemas individuais, cada qual com complexidade muito menor do que a complexidade do modelo que representa sistema como um todo.

3. Teoria de Controle Supervisório

A Teoria de Controle Supervisório constitui-se como uma importante ferramenta para o controle de sistemas a eventos discretos. Baseado em um modelo que descreve o comportamento livre do sistema a ser controlado e num conjunto de especificações comportamentais é possível realizar a síntese formal de um supervisor. A ação de controle do supervisor restringe, de forma minimamente restritiva, o comportamento do sistema de forma a satisfazer o conjunto de especificações.

Na seção inicial deste capítulo são apresentados os conceitos fundamentais da Teoria de Controle Supervisório. Na seção seguinte é apresentada a abordagem Modular Local desta teoria. Esta abordagem prevê que seja realizada a síntese de um conjunto de supervisores, sendo que cada supervisor garante a satisfação de um subconjunto das especificações comportamentais. Ao final do capítulo é apresentado um exemplo que ilustra a aplicação da Teoria de Controle Supervisório. Este exemplo também será empregado para ilustrar o método de implementação a ser apresentado nos Capítulos 5 e 7.

3.1 Conceituação geral

No capítulo anterior foi introduzido um formalismo que permite representar o comportamento livre de um sistema a eventos discretos. Este comportamento é descrito pela linguagem gerada e pela linguagem marcada por um autômato e deve corresponder a todas as possíveis seqüências de eventos que podem ser observadas neste sistema segundo a perspectiva empregada para interpretação do mesmo. Algumas destas seqüências de eventos podem, entretanto, descrever um comportamento inadequado do sistema. Como exemplos de comportamento inadequado podem-se citar o início de dois processos concorrentes que ao longo do desenvolvimento necessitam realizar a alocação de recursos comuns, o que pode conduzir o sistema a uma situação de bloqueio, onde cada um destes processos não pode ser concluído, pois necessita de recursos que já estão alocados pelo outro processo; o deslocamento de dois robôs manipuladores com trajetórias que podem se interceptar e resultar na colisão dos mesmos; a produção de um

determinado tipo de peça para a qual não há demanda em detrimento de um outro tipo de elevada demanda.

De forma geral, o comportamento desejado do sistema é determinado por um conjunto de especificações que consideram aspectos como segurança e vivacidade do sistema, prioridade de eventos e justiça na alocação de recursos. No caso de sistemas compostos, as especificações comportamentais estabelecem, também, o acoplamento dos diversos subsistemas, bem como, a coordenação da operação concorrente dos mesmos. Conforme QUEIROZ (2004), uma especificação de segurança visa garantir que "nada de ruim aconteça" ao sistema, ao passo que uma especificação de vivacidade visa garantir que "algo de bom aconteça". Em geral, o grau de abstração empregado na representação do comportamento livre do sistema é influenciado pela formulação e representação das especificações. Assim como o modelo que descreve o comportamento livre do sistema, tais especificações são representadas através de autômatos.

A Teoria de Controle Supervisório (RAMADGE e WONHAN, 1987) propõe que uma estrutura denominada supervisor observe a seqüência de eventos gerados espontaneamente no sistema a ser controlado e que, instantaneamente após a observação de um novo evento, determine o subconjunto de eventos que concatenados a esta seqüência preservam o comportamento do sistema dentro do desejado. Este subconjunto de eventos define uma entrada de controle e especifica todos os eventos que estão habilitados a ocorrer. Os eventos que não estão habilitados são impedidos de ocorrer. Diz-se que a ação de controle do supervisor é de natureza permissiva, pois a escolha de qual evento irá ocorrer, dentre os habilitados, cabe ao próprio sistema. Após a observação de um novo evento a entrada de controle deve ser atualizada com um novo subconjunto de eventos a serem habilitados.

Conforme (WONHAM, 2003) um supervisor corresponde a um mapeamento que leva da linguagem gerada a um subconjunto de eventos a serem habilitados. Este mapeamento pode ser denotado como $\mathcal{V}: L(\mathbf{G}) \rightarrow 2^{\Sigma}$, onde **G** é o autômato que representa o comportamento livre do sistema a ser controlado e Σ o alfabeto de eventos empregados na representação de **G**. Para uma dada seqüência de eventos *s*, a entrada de controle é então representada por $\mathcal{V}(s)$.

QUEIROZ (2004) representa de forma dual a ação do supervisor, especificando o conjunto mínimo de eventos a serem desabilitados após a ocorrência de uma dada seqüência de eventos. Este conjunto é determinado retirando-se do conjunto de eventos que são fisicamente possíveis após a ocorrência da seqüência em questão todos os eventos habilitados pelo supervisor. Pode-se representar este conjunto como $\Gamma^{G}(\delta^{G}(q0^{G}, s)) - \mathcal{V}(s)$. Ao supervisor habilitador de eventos há um supervisor desabilitador de eventos correspondente, tal que a ação de controle de ambos restringe

de forma equivalente o comportamento do sistema. Sendo $\mathfrak{S}$ o supervisor desabilitador, o conjunto de eventos desabilitados por este supervisor é representado por $\mathfrak{S}(s)$.

Na Figura 3.1(a) é representada graficamente a arquitetura de controle supervisório com ação de controle conforme definido em (WONHAM, 2003). Na Figura 3.1(b) apresenta-se a arquitetura de controle supervisório dual.

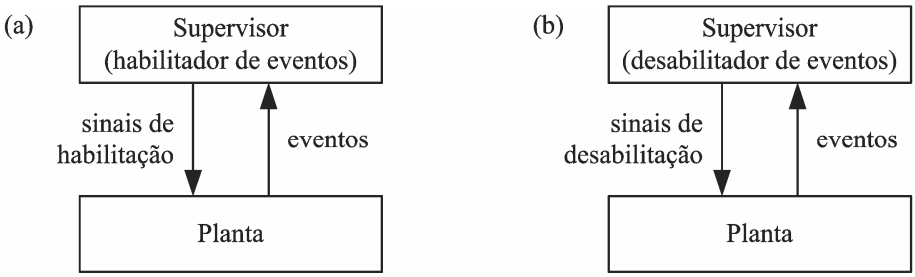


Figura 3.1 – Arquitetura de Controle Supervisório

Seja **G** um autômato que descreve o comportamento livre do sistema a ser controlado e $\mathfrak{S}$ um supervisor que restringe este comportamento. O comportamento do sistema obtido sob ação de controle do supervisor corresponde ao par de linguagens $L(\mathfrak{S}/\mathbf{G})$ e $L_m(\mathfrak{S}/\mathbf{G})$, onde $\mathfrak{S}/\mathbf{G}$ é o autômato que gera e marca estas linguagens.

A linguagem gerada sob supervisão representa todas as possíveis seqüências de eventos que pertencem à linguagem gerada por **G** e que podem ser geradas sob ação de supervisão, sendo definida recursivamente como:

i) $\varepsilon \in L(\mathfrak{S}/\mathbf{G})$;

ii) $[s \in L(\mathfrak{S}/\mathbf{G})] \wedge [s\sigma \in L(\mathbf{G})] \wedge [\sigma \notin \mathfrak{S}(s)] \leftrightarrow [s\sigma \in L(\mathfrak{S}/\mathbf{G})]$.

RAMADGE e WONHAM (1987) definem uma terceira linguagem, a qual é composta pelas palavras que o autômato **G** reconhece como palavras que conduzem à conclusão de uma tarefa e que podem ser geradas sob supervisão. Esta linguagem é obtida por $L_m(\mathbf{G}) \cap L(\mathfrak{S}/\mathbf{G})$.

A Teoria de Controle Supervisório prevê que o supervisor, além de restringir a linguagem gerada pelo sistema a ser controlado, pode alterar a definição de tarefa concluída. Assim, é possível que parte das palavras em $L_m(\mathbf{G}) \cap L(\mathfrak{S}/\mathbf{G})$ não sejam consideradas como palavras que conduzem a conclusão de qualquer tarefa sob supervisão, neste caso o supervisor é dito ser marcador. Alguns autores, tais como QUEIROZ (2004), denominam este tipo de supervisor como desmarcador. Seja $M \subseteq L_m(\mathbf{G})$ o conjunto de palavras que o supervisor reconhece como palavras que conduzem à

conclusão de uma tarefa. A linguagem marcada sob supervisão é dada por $L_m(\mathfrak{S}/\mathbf{G}) = M \cap L(\mathfrak{S}/\mathbf{G})$. Para WONHAM (2003) o supervisor é não-bloqueante se $\overline{L_m(\mathfrak{S}/\mathbf{G})} = L(\mathfrak{S}/\mathbf{G})$.

A Teoria de Controle Supervisório prevê a distinção dos eventos que podem ser desabilitados pela ação do supervisor daqueles em que esta ação não é factível. Isto é associado ao conceito de controlabilidade de eventos e linguagens.

Para realizar a síntese do supervisor, o alfabeto de eventos deve ser particionado em eventos controláveis, aqueles cuja ocorrência pode ser evitada através da desabilitação, e eventos não-controláveis, aqueles cuja ocorrência não pode ser impedida por um sistema de controle externo. Se o alfabeto de eventos é representado por Σ, o alfabeto de eventos controláveis pode ser representado por Σ_c e o alfabeto de eventos não-controláveis por Σ_{uc}, sendo que $\Sigma = \Sigma_c \cup \Sigma_{uc}$ e $\Sigma_c \cap \Sigma_{uc} = \varnothing$. Tipicamente o início de um processo ou de uma operação é associado a um evento controlável, ao passo que a conclusão de operações ou processos, a ocorrência de pane na operação, a sinalização de informações através de sensores, ou ainda, ações executadas por um operador humano são associadas a eventos não-controláveis.

Em um diagrama de transição de estados a controlabilidade de eventos é usualmente especificada através do emprego de dois tipos de arcos. Um arco interceptado por uma pequena reta é utilizado para representar uma transição de estado associada a um evento controlável e um arco simples é empregado para representar uma transição associada a um evento não-controlável.

Seja **G** o autômato que descreve o comportamento livre do sistema a ser controlado, e Σ o conjunto de eventos empregado na representação deste autômato. A linguagem $K \subseteq \Sigma^*$ é dita ser controlável em relação a $L(\mathbf{G})$ e Σ_{uc}, ou simplesmente **G**-controlável, se e somente se após qualquer seqüência de eventos que pertença ao prefixo-fechamento desta linguagem ($s \in \overline{K}$) a ocorrência de um evento não-controlável ($\sigma \in \Sigma_{uc}$) que seja prevista pelo autômato **G** ($s\sigma \in L(\mathbf{G})$) preserva a seqüência assim formada dentro do prefixo-fechamento desta linguagem ($s\sigma \in \overline{K}$). Ou seja, a linguagem K é **G**-controlável se e somente se $\overline{K}\,\Sigma_{uc} \cap L(\mathbf{G}) \subseteq \overline{K}$.

Em particular a linguagem vazia, $L(\mathbf{G})$ e Σ^* são controláveis em relação à $L(\mathbf{G})$ e Σ_{uc}. Além disto, K é **G**-controlável se e somente se $\overline{K}$ é **G**-controlável.

Na Figura 3.2 é representada através de um diagrama de Venn a relação das linguagens $\overline{K} \subset L(\mathbf{G}) \subset \Sigma^*$. É representada também a seqüência de eventos $s \in \overline{K}$. A concatenação do evento não-controlável σ_1 à palavra s mantém a palavra assim obtida ($s\sigma_1$) dentro da linguagem $\overline{K}$, já a concatenação do evento não-controlável σ_2 conduz a palavra assim obtida ($s\sigma_2$) para fora da

linguagem $L(\mathbf{G})$. Nestes dois casos não há violação da controlabilidade da linguagem $\overline{K}$. Por outro lado, a concatenação do evento não-controlável σ_3 à palavra s conduz a palavra assim obtida ($s\sigma_3$) para fora da linguagem $\overline{K}$, porém, preserva dentro da linguagem $L(\mathbf{G})$. Assim ($s\sigma_3 \in \overline{K}\,\Sigma_{uc} \cap L(\mathbf{G})$, porém, ($s\sigma_3 \notin \overline{K}$). O que viola a controlabilidade da linguagem $\overline{K}$.

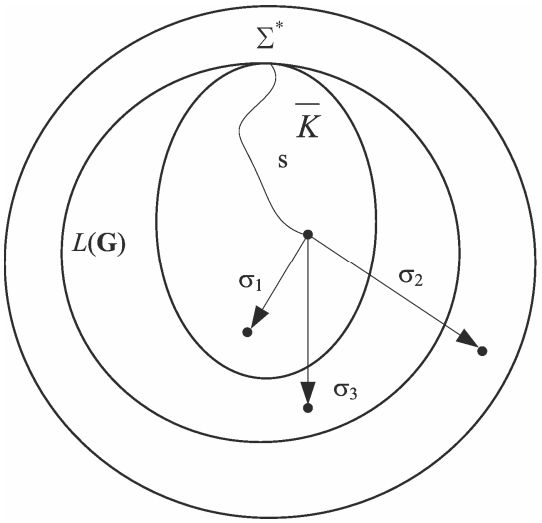


Figura 3.2 – Controlabilidade de linguagem

RAMADGE e WONHAN (1987) definem $\mathfrak{C}(K) = \{K_i \subseteq K \mid K_i$ é **G**-controlável$\}$ como o conjunto de todas as sublinguagens de K que são **G**-controláveis. De acordo com estes autores, a classe de linguagens assim definida é não-vazia, pois $\varnothing \in \mathfrak{C}(K)$. Além disto, esta classe de linguagens é fechada para união, isto é, se $K_1 \in \mathfrak{C}(K)$ e $K_2 \in \mathfrak{C}(K)$ então $K_1 \cup K_2 \in \mathfrak{C}(K)$. Por conseqüência, esta classe de linguagens possui um elemento supremo, denotado por $\sup\mathfrak{C}(K)$, o qual contém todas as linguagens pertencentes à $\mathfrak{C}(K)$.

Em (WONHAM e RAMADGE, 1987) é apresentado um algoritmo que permite determinar $\sup\mathfrak{C}(K)$ cuja complexidade é polinomial no número de estados do autômato **G** e do autômato que marca a linguagem K.

Conforme o teorema 6.1 (*i*) apresentado em (RAMAGE e WONHAN, 1987):

Seja $K \subseteq L_m(\mathbf{G})$, com $K \neq \varnothing$.

*Existe um supervisor $\mathcal{S}$ não-bloqueante (possivelmente marcador) para **G** tal que $L_m(\mathcal{S}/\mathbf{G}) = K$ se e somente se K é controlável em relação à $L(\mathbf{G})$ e Σ_{uc}.*

Seja K a linguagem que marca o comportamento desejado para o sistema a ser controlado, e **G** o autômato que representa o comportamento livre deste sistema. Caso esta linguagem não seja

G-controlável, então, o comportamento ótimo sob supervisão para este sistema é $L_m(\mathfrak{S}/\mathbf{G}) = \sup\mathfrak{C}(K)$.

A definição de supervisor como apresentado no início desta seção não é prática do ponto de vista da realização do mesmo, pois não é factível executar o mapeamento de uma linguagem infinita. Um possível procedimento para realizar um supervisor é através de um par $\mathfrak{S} = (\mathbf{S}, \Phi)$, onde $\mathbf{S} = (Q^S, \Sigma^S, \delta^S, q0^S, Q_m{}^S)$ é um autômato (com $\Sigma^S = \Sigma^G = \Sigma$) e $\Phi : Q^S \rightarrow 2^\Sigma$ é uma função que mapeia cada estado do autômato **S** em um subconjunto de eventos a serem desabilitados. Esta função é denominada mapa de saída do supervisor $\mathfrak{S}$.

De acordo com a proposição 3.6.2 apresentada em (WONHAM, 2003):

*Seja $\mathfrak{S}$ um supervisor, possivelmente marcador, e **S** um autômato, tal que $L_m(\mathfrak{S}/\mathbf{G}) = L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \neq \varnothing$. Se forem satisfeitas as propriedades (i) a (iii) apresentadas abaixo, então o autômato **S** realiza o supervisor $\mathfrak{S}$, tal que $L_m(\mathfrak{S}/\mathbf{G}) = L_m(\mathbf{S}) \cap L_m(\mathbf{G})$ e $L(\mathfrak{S}/\mathbf{G}) = L(\mathbf{S}) \cap L(\mathbf{G})$.*

*i) **S** é um autômato TRIM (acessível e coacessível);*

*ii) $L_m(\mathbf{S})$ é **G**-controlável;*

iii) as linguagens $L_m(\mathbf{S})$ e $L_m(\mathbf{G})$ são não-conflitantes, isto é, $\overline{L_m(\mathbf{S}) \cap L_m(\mathbf{G})} = L(\mathbf{S}) \cap L(\mathbf{G})$.

Conforme QUEIROZ (2004) dois supervisores, $\mathfrak{S}_1 = (\mathbf{S_1}, \Phi_1)$ e $\mathfrak{S}_2 = (\mathbf{S_2}, \Phi_2)$, são equivalentes se suas ações de controle sobre o sistema a ser controlado produzirem o mesmo comportamento sob supervisão, ou seja, $L_m(\mathfrak{S}_1/\mathbf{G}) = L_m(\mathfrak{S}_2/\mathbf{G})$ e $L(\mathfrak{S}_1/\mathbf{G}) = L(\mathfrak{S}_2/\mathbf{G})$.

Em particular, o autômato $\mathfrak{S}/\mathbf{G}$ pode ser empregado na realização do supervisor $\mathfrak{S}$ para controle do sistema representado por **G**, onde $L_m(\mathfrak{S}/\mathbf{G})$ é o comportamento ótimo sob supervisão. Entretanto, o autômato $\mathfrak{S}/\mathbf{G}$ contém informações redundantes, já consideradas na estrutura do autômato **G**.

Sejam os supervisores $\mathfrak{S} = (\mathfrak{S}/\mathbf{G}, \Phi)$ e $\mathfrak{S}' = (\mathbf{S}', \Phi')$ empregados no controle do sistema **G**, onde $\mathfrak{S}/\mathbf{G}$ é o autômato que gera e marca o comportamento de **G** sob supervisão de $\mathfrak{S}$. O supervisor $\mathfrak{S}'$ é uma representação reduzida do supervisor $\mathfrak{S}$ se ambos forem equivalentes e se o número de estados do autômato **S'** for menor do que o número de estados do autômato $\mathfrak{S}/\mathbf{G}$. VAZ e WONHAM (1986) provam que existe um valor mínimo para o número de estados do autômato que pode representar o supervisor $\mathfrak{S}$, porém este autômato não é único. Os referidos autores definem um algoritmo que determina um autômato que é mínimo no número de estados e que pode ser

utilizado para representar o supervisor $\mathcal{S}$. A complexidade de tal algoritmo é exponencial no número de estados do autômato $\mathcal{S}$/**G**. SU e WONHAM (2004) e MINHAS (2002) apresentam algoritmos com complexidade polinomial no número de estados do autômato $\mathcal{S}$/**G** que determinam um autômato com um menor número de estados do que autômato $\mathcal{S}$/**G** para representação do supervisor $\mathcal{S}$.

3.2 Abordagem Modular Local

Considere um sistema composto e um conjunto de especificações comportamentais. Uma possível abordagem para este problema consiste em obter um autômato que descreve o comportamento do sistema e outro que representa o conjunto de especificações como um todo. De posse destes autômatos é realizada a síntese de um supervisor global que coordena o comportamento do sistema de forma a satisfazer simultaneamente todas as especificações.

A síntese de um supervisor global (abordagem monolítica conforme descrita no parágrafo anterior) apresenta as seguintes limitações:

i) a complexidade computacional para síntese do mesmo é exponencialmente proporcional ao número de subsistemas envolvidos e ao número de especificações (QUEIROZ, 2000);

ii) o número de estados do autômato que marca o comportamento ótimo sob supervisão é exponencialmente proporcional ao número de subsistemas envolvidos e ao número de especificações, o que pode inviabilizar: a síntese do supervisor, a interpretação do comportamento obtido sob supervisão; a obtenção de uma representação reduzida do supervisor, bem como, a implementação do supervisor em um dispositivo de controle;

iii) a inclusão ou exclusão de um subsistema ou de uma especificação requer que os procedimentos de síntese, redução e implementação do supervisor sejam refeitos na íntegra.

A aplicação da abordagem Modular Local da Teoria de Controle Supervisório (QUEIROZ, 2000) (QUEIROZ e CURY, 2000a, 2000b, 2002a) requer que o sistema composto seja descrito por uma representação por sistema produto $\{\mathbf{G_i} \mid i \in I = \{1, ..., n\}\}$, com $\mathbf{G_i} = (Q^{Gi}, \Sigma^{Gi}, \delta^{Gi}, q0^{Gi}, Q_m^{Gi})$. O conjunto total de eventos é obtido por $\Sigma = Y_{i=1}^{n} \Sigma^{Gi}$. Os conjuntos de eventos controláveis e de eventos não-controláveis do subsistema $\mathbf{G_i}$ são, respectivamente, representados por Σ_c^{Gi} e Σ_{uc}^{Gi}.

Considere o conjunto de autômatos representando cada uma das especificações comportamentais $\{\mathbf{E_j} \mid j \in J = \{1, ..., m\}\}$, com $\mathbf{E_j} = (Q^{Ej}, \Sigma^{Ej}, \delta^{Ej}, q0^{Ej}, Q_m^{Ej})$. Para cada especificação, deve ser obtida a planta local correspondente, designadas $\mathbf{G_{lj}}$. Isto é realizado através

do produto síncrono de todos os autômatos em $\{\mathbf{G_i} \mid i \in I\}$ que compartilham algum evento com a especificação em questão, isto é, os autômatos em que $\Sigma^{Ej} \cap \Sigma^{Gi} \neq \varnothing$.

A abordagem Modular Local prevê que seja realizada a síntese de um supervisor para cada uma das especificações comportamentais. Além disto, a síntese de cada um destes supervisores considera apenas os subsistemas que definem a planta local correspondente. Os supervisores assim sintetizados são denominados supervisores locais. Se pelo menos um supervisor local estabelecer a desabilitação de um determinado evento, então a ocorrência deste evento estará desabilitada pela ação conjunta dos supervisores.

Seja $\{\mathfrak{S}/\mathbf{G_j} \mid j \in J\}$ o conjunto de autômatos que representam o comportamento ótimo da planta local $\mathbf{G_{lj}}$ sob ação de supervisão do supervisor local $\mathfrak{S}_j$ correspondente. O conjunto de supervisores $\{\mathfrak{S}_j \mid j \in J\}$ é dito ser modular local se:

$$\overline{\|_{j=i}^{m} L_m(\mathfrak{S}/\mathbf{G}_j)} = \|_{j=1}^{m} \overline{L_m(\mathfrak{S}/\mathbf{G}_j)}$$

Caso seja verificada a Modularidade Local do conjunto de supervisores locais, então, o Teorema 7 apresentado em (QUEIROZ, 2000) garante que a ação conjunta dos supervisores locais é equivalente à ação do supervisor global que atende a todas as especificações simultaneamente. Caso esta propriedade não seja verificada, significa que há conflito entre os mesmos, e o emprego deste conjunto de supervisores locais no controle do sistema poderá conduzir o mesmo a uma situação de bloqueio, o que não é aceitável para o caso geral. Em (QUEIROZ, 2000) encontra-se uma seção que aborda a resolução de conflito entre supervisores.

Denomina-se célula de controle ao conjunto formado por um supervisor local e os subsistemas que definem a planta local correspondente.

A abordagem Modular Local apresenta as seguintes vantagens sobre a abordagem monolítica:

i) a complexidade computacional para síntese do conjunto de supervisores locais é significativamente reduzida em relação à complexidade computacional para síntese do supervisor global (QUEIROZ, 2000);

ii) como o autômato que representa o comportamento ótimo de cada planta local sob ação do supervisor local correspondente apresenta um menor número de estados do que o autômato que representa o comportamento ótimo da planta global sob ação do supervisor global, viabiliza-se: a interpretação destes comportamentos; a obtenção de uma representação reduzida dos supervisores locais, bem como, a implementação dos mesmos em um dispositivo de controle;

iii) com a inclusão ou exclusão de um subsistema, apenas os supervisores locais que compartilham algum evento com o subsistema em questão deverão ser sintetizados novamente, os demais permanecerão inalterados;

iv) a inclusão de uma nova especificação não altera os supervisores locais já sintetizados;

v) a remoção de uma especificação implica na exclusão do supervisor local correspondente, porém não afeta os demais supervisores;

vi) preserva a modularidade natural do sistema e das especificações comportamentais.

A principal limitação deste procedimento consiste na verificação da Modularidade Local dos Supervisores, a qual apresenta complexidade computacional elevada sendo, contudo, menor ou igual à complexidade computacional para síntese do supervisor global (QUEIROZ, 2000).

3.3 Um exemplo motivador

Esta seção apresenta um exemplo que permite ilustrar o procedimento de modelagem de sistemas a eventos discretos e de síntese de supervisores. Este exemplo também será empregado para ilustrar o método de implementação a ser apresentado nos Capítulos 5 e 7.

A célula de manufatura apresentada na Figura 3.3 é constituída de uma mesa rotativa com quatro posições de trabalho (M0); um sistema de classificação e transporte de peças (M1); um sistema de furação (M2); um sistema de teste (M3); um manipulador robótico (M4) e um dispositivo de alimentação (M5).

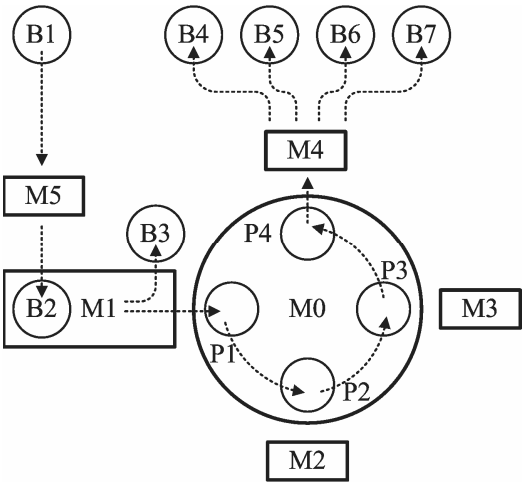


Figura 3.3 – Célula de manufatura

Esta célula de manufatura opera com três tipos diferentes de peças, as quais apresentam dimensões variáveis. Estas peças são armazenadas de forma aleatória no armazém de peças brutas (B1). O dispositivo de alimentação retira as peças deste armazém e as transfere para o sistema de classificação e transporte (na posição de trabalho B2). Em função do resultado da classificação, ou as peças são aceitas e transferidas para a mesa rotativa (posição P1), ou são descartadas no armazém intermediário (B3). Uma peça, de um determinado tipo, é rejeitada sempre que a quantidade de peças já processadas mais aquelas que estão em processamento iguala a quantidade máxima a ser produzida deste tipo de peças. Uma peça também é rejeitada caso suas dimensões estejam fora de limites pré-estabelecidos. As peças aceitas são furadas (posição P2) e testadas (posição P3). Há dois procedimentos de teste. O procedimento de teste T_A deve ser realizado sempre que a peça foi furada com sucesso. O procedimento de teste T_B deve ser realizado caso ocorra falha durante o processo de furação. As peças são, então, retiradas da mesa (posição P4) e armazenadas nos armazéns B4 a B7. Há um armazém para armazenagem de cada tipo específico de peça e outro para as peças rejeitadas no teste.

Em QUEIROZ e CURY (2002b) é abordada uma versão simplificada desta célula de manufatura. A modelagem do sistema e das especificações de controle apresentadas na referida obra foram empregadas como referência para a modelagem apresentada a seguir.

A célula de manufatura foi modelada empregando a representação por sistema produto $\{\mathbf{G_i} \mid i \in I\}$ com $I = \{0, \ldots, 5\}$. A Figura 3.4 apresenta o conjunto de autômatos empregados para representar o comportamento independente de cada subsistema e a Tabela 3.1 apresenta a semântica dos estados e eventos correspondentes.

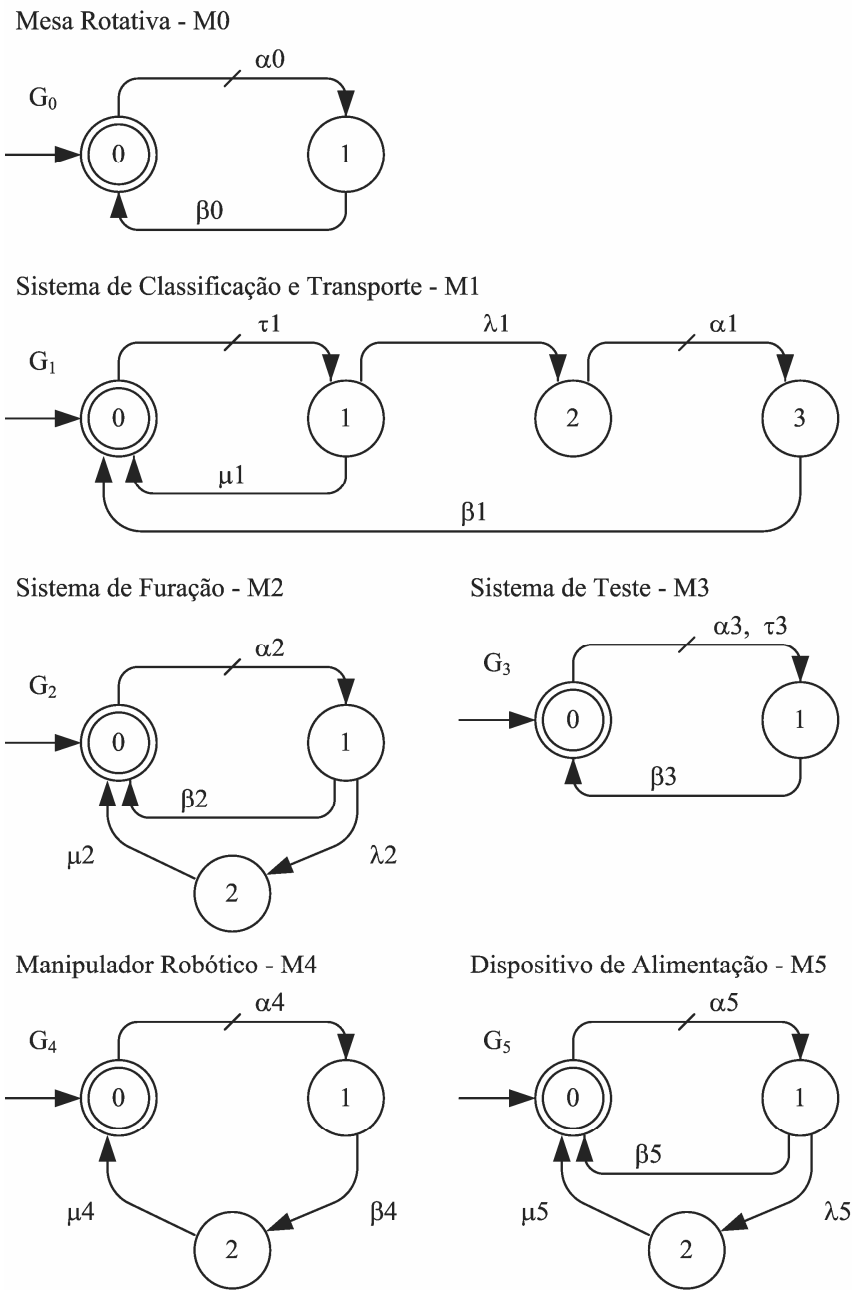


Figura 3.4 – Conjunto de autômatos representando o comportamento independente dos subsistemas que constituem a célula de manufatura

Tabela 3.1 – Semântica de estados e eventos		
autômato	estado	evento
G_0	0 - repouso	α0 - iniciar rotação de 90°
	1 - rotacionando	β0 - rotação concluída
G_1	0 - repouso	τ1 - iniciar classificação
	1 - classificando	λ1 - classificada e aceita
	2 - em pausa	μ1 - classificada e rejeitada
	3 - transportando	α1 - iniciar transporte
		β1 - transportada
G_2	0 - repouso	α2 - iniciar furação
	1 - furando	β2 - furada com sucesso
	2 - em pane	λ2 - pane na furação
		μ2 - sistema de furação reparado
G_3	0 - repouso	α3 - iniciar procedimento de teste T_A
	1 - testando	τ3 - iniciar procedimento de teste T_B
		β3 - teste concluído
G_4	0 - repouso	α4 - iniciar retirada e armazenagem
	1 - retirando da mesa	β4 - retirada da mesa
	2 - armazenando	μ4 - armazenada
G_5	0 - repouso	α5 - iniciar alimentação da célula
	1 - alimentando a célula	β5 - célula alimentada
	2 - sem peças para alimentação	λ5 - armazém B1 vazio
		μ5 - armazém B1 recarregado

O comportamento desejado é estabelecido através do conjunto de especificações comportamentais $\{\mathbf{E_j} \mid j \in J\}$, com J = {a, b1, b2, b3, b4, c1, c2, c3, d}.

E_a - evitar rotacionar a mesa sem pelo menos, ou uma peça bruta na posição P1, ou uma peça furada na posição P2, ou uma peça testada na posição P3;

E_{b1} - evitar rotacionar a mesa e transportar uma peça bruta simultaneamente;

E_{b2} - evitar rotacionar a mesa e furar simultaneamente;

E_{b3} - evitar rotacionar a mesa e testar simultaneamente;

E_{b4} - evitar rotacionar a mesa e retirar peça simultaneamente;

E_{c1} - evitar:

i) duas ou mais peças na posição P1;

ii) furar sem uma peça bruta em P2;

iii) rotacionar com uma peça não furada na posição P2;

E_{c2} - evitar:

i) furar peça já furada;

ii) testar sem uma peça furada na posição P3;

iii) rotacionar com uma peça não testada na posição P3;

iv) testar com procedimento T_A uma peça que resulta de uma furação em que ocorreu pane;

v) testar com procedimento T_B uma peça que resulta de uma furação concluída com sucesso;

E_{c3} - evitar:

i) testar peça já testada;

ii) retirar peça quando não há peça na posição P4;

iii) rotacionar com peça na posição P4;

E_d - evitar:

i) under-flow e over-flow do armazém B2;

ii) alimentar e classificar ou transportar simultaneamente;

A representação do comportamento independente dos subsistemas (Figura 3.4) que compõem a célula de manufatura foi realizada num nível de abstração que considera apenas os eventos necessários para expressar a coordenação dos mesmos. Além disto, o conjunto de especificações comportamentais apresentadas acima não estabelece como realizar o armazenamento das peças através do manipulador robótico nem, tão pouco, como o sistema de classificação e transporte estabelece a rejeição de peças. No Capítulo 4 são discutidos os principais aspectos a serem considerados na implementação do controle, é então introduzido o conceito de procedimento operacional. A associação de procedimentos operacionais aos eventos empregados na representação do comportamento independente dos diversos sistemas permite realizar o detalhamento das abstrações adotadas na etapa de modelagem. Os procedimentos operacionais também permitem implementar ações de controle que podem ser estabelecidas trivialmente de forma empírica sem a necessidade da adoção de uma representação mais refinada do

comportamento independente de cada subsistema e da aplicação da Teoria de Controle Supervisório. O Exemplo 5.7 apresenta como é realizada a rejeição e armazenamento de peças.

Considerando o conjunto de autômatos apresentados na Figura 3.4 e as especificações comportamentais apresentadas acima, a aplicação da abordagem modular local resulta no conjunto de supervisores locais $\{\mathcal{S}_j \mid j \in J\}$, com J = {a, b1, b2, b3, b4, c1, c2, c3, d}. A Figura 3.5 apresenta o conjunto de autômatos empregados em uma representação reduzida de tais supervisores, e a Tabela 3.2 os mapas de saída correspondentes. A Tabela 3.3 apresenta, para cada supervisor local, a planta local e a célula de controle correspondentes. A quarta coluna desta tabela apresenta o número de estados e o número de transições do autômato que representa o comportamento ótimo sob supervisão da respectiva planta local sob ação de supervisão, ou seja, o autômato que representa de forma não-reduzida o supervisor local. A quinta coluna apresenta o número de estados e o número de transições do autômato empregado na representação reduzida do supervisor local correspondente apresentados na Figura 3.5.

Caso fosse adotada a abordagem monolítica para síntese do supervisor global, seria necessário realizar o produto síncrono dos autômatos apresentados na Figura 3.4. O autômato assim obtido representa o comportamento da planta global e possui 432 estados e 3.204 transições. Verifica-se que o somatório do número de estados e transições dos autômatos empregados para representar o comportamento independente dos subsistemas que constituem a célula de manufatura (Figura 3.4) resulta 17 estados e 21 transições. Para representar de forma não-reduzida o supervisor global seria necessário um autômato com 2.082 estados e 6.914 transições. Uma possível representação reduzida para o supervisor global seria através de um autômato com 362 estados e 2.442 transições. Da Tabela 3.3, verifica-se que o somatório do número de estados e transições do conjunto de autômatos empregados na representação reduzida dos supervisores locais resulta 29 estados e 68 transições.

Após a síntese do conjunto de nove supervisores locais foi realizado o teste de modularidade através da realização do produto síncrono dos autômatos que representam o comportamento ótimo de cada planta local sob ação do supervisor local correspondente. Isto resultou em um autômato idêntico ao autômato que representa o comportamento ótimo da planta global sob ação do supervisor global, ou seja, o autômato que representa de forma não-reduzida o supervisor global.

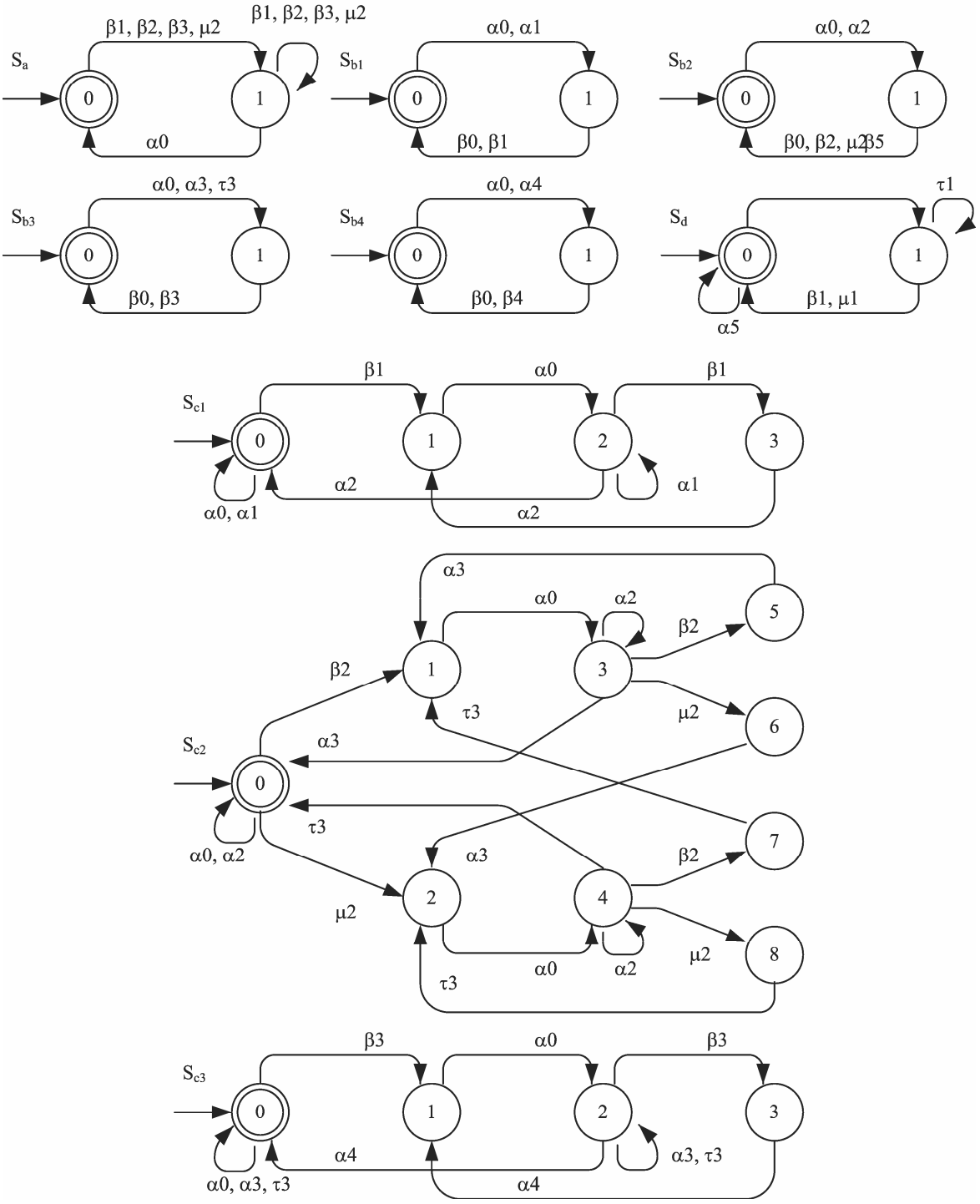


Figura 3.5 – Conjunto de autômatos empregados na representação reduzida dos supervisores locais

Tabela 3.2 – Mapa de saída da representação reduzida dos supervisores		
supervisor local	mapa de saída	
$\mathcal{S}_a$	$\Phi_a(0) = \{\alpha 0\};$	$\Phi_a(1) = \varnothing$
$\mathcal{S}_{b1}$	$\Phi_{b1}(0) = \varnothing;$	$\Phi_{b1}(1) = \{\alpha 0, \alpha 1\}$
$\mathcal{S}_{b2}$	$\Phi_{b2}(0) = \varnothing;$	$\Phi_{b2}(1) = \{\alpha 0, \alpha 2\}$
$\mathcal{S}_{b3}$	$\Phi_{b3}(0) = \varnothing;$	$\Phi_{b3}(1) = \{\alpha 0, \alpha 3, \tau 3\}$
$\mathcal{S}_{b4}$	$\Phi_{b4}(0) = \varnothing;$	$\Phi_{b4}(1) = \{\alpha 0, \alpha 4\}$
$\mathcal{S}_{c1}$	$\Phi_{c1}(0) = \{\alpha 2\};$ $\Phi_{c1}(1) = \{\alpha 1, \alpha 2\};$	$\Phi_{c1}(2) = \{\alpha 0\};$ $\Phi_{c1}(3) = \{\alpha 0, \alpha 1\}$
$\mathcal{S}_{c2}$	$\Phi_{c2}(0) = \{\alpha 3, \tau 3\};$ $\Phi_{c2}(1) = \Phi_{c2}(2) = \{\alpha 2, \alpha 3, \tau 3\};$ $\Phi_{c2}(3) = \{\alpha 0, \tau 3\};$	$\Phi_{c2}(4) = \{\alpha 0, \alpha 3\};$ $\Phi_{c2}(5) = \Phi_{c2}(6) = \{\alpha 0, \alpha 2, \tau 3\};$ $\Phi_{c2}(7) = \Phi_{c2}(8) = \{\alpha 0, \alpha 2, \alpha 3\};$
$\mathcal{S}_{c3}$	$\Phi_{c3}(0) = \{\alpha 4\};$ $\Phi_{c3}(1) = \{\alpha 3, \tau 3, \alpha 4\};$	$\Phi_{c3}(2) = \{\alpha 0\};$ $\Phi_{c3}(3) = \{\alpha 0, \alpha 3, \tau 3\}$
$\mathcal{S}_d$	$\Phi_d(0) = \{\tau 1\};$	$\Phi_d(1) = \{\alpha 5\}$

Tabela 3.3 – Resultados da aplicação da abordagem modular local				
supervisor local	planta local	célula de controle	representação não-reduzida (est. x trans.)	representação reduzida (est. x trans.)
$\mathcal{S}_a$	$\mathbf{Gl_a = G_0 \parallel G_1 \parallel G_2 \parallel G_3}$	$CC_a = \{\mathcal{S}_a, \mathbf{G_0}, \mathbf{G_1}, \mathbf{G_2}, \mathbf{G_3}\}$	96x464	2x9
$\mathcal{S}_{b1}$	$\mathbf{Gl_{b1} = G_0 \parallel G_1}$	$CC_{b1} = \{\mathcal{S}_{b1}, \mathbf{G_0}, \mathbf{G_1}\}$	7x14	2x4
$\mathcal{S}_{b2}$	$\mathbf{Gl_{b2} = G_0 \parallel G_2}$	$CC_{b2} = \{\mathcal{S}_{b2}, \mathbf{G_0}, \mathbf{G_2}\}$	4x6	2x5
$\mathcal{S}_{b3}$	$\mathbf{Gl_{b3} = G_0 \parallel G_3}$	$CC_{b3} = \{\mathcal{S}_{b3}, \mathbf{G_0}, \mathbf{G_3}\}$	3x5	2x5
$\mathcal{S}_{b4}$	$\mathbf{Gl_{b4} = G_0 \parallel G_4}$	$CC_{b4} = \{\mathcal{S}_{b4}, \mathbf{G_0}, \mathbf{G_4}\}$	5x8	2x4
$\mathcal{S}_{c1}$	$\mathbf{Gl_{c1} = G_0 \parallel G_1 \parallel G_2}$	$CC_{c1} = \{\mathcal{S}_{c1}, \mathbf{G_0}, \mathbf{G_1}, \mathbf{G_2}\}$	84x257	4x8
$\mathcal{S}_{c2}$	$\mathbf{Gl_{c2} = G_0 \parallel G_2 \parallel G_3}$	$CC_{c2} = \{\mathcal{S}_{c2}, \mathbf{G_0}, \mathbf{G_2}, \mathbf{G_3}\}$	60x138	9x18
$\mathcal{S}_{c3}$	$\mathbf{Gl_{c3} = G_0 \parallel G_3 \parallel G_4}$	$CC_{c3} = \{\mathcal{S}_{c3}, \mathbf{G_0}, \mathbf{G_3}, \mathbf{G_4}\}$	36x93	4x10
$\mathcal{S}_d$	$\mathbf{Gl_d = G_0 \parallel G_5}$	$CC_d = \{\mathcal{S}_d, \mathbf{G_0}, \mathbf{G_5}\}$	7x9	2x5

Conforme será observado no Capítulo 5, a memória de CLP necessária para realizar a implementação do controle é proporcional ao somatório de estados do modelo empregado para representar o comportamento livre do sistema a ser controlado, bem como, do somatório de estados do conjunto de autômatos empregados na representação do conjunto de supervisores. Deste fato, e da comparação dos valores apresentados nos parágrafos anteriores, verifica-se a importância da adoção da abordagem Modular Local na síntese dos supervisores associado a procedimentos de redução de supervisores.

O método de implementação proposto nesta Tese foi utilizado para realizar, com sucesso, o controle da célula de manufatura apresentada nesta seção. Na Seção 5.9 e no Exemplo 7.1 são descritos os controladores utilizados na implementação concentrada e distribuída do controle.

3.4 Conclusão

Visando o desenvolvimento de um sistema de controle supervisório, a Teoria de Controle Supervisório permite a síntese formal de supervisores que garantem que o comportamento de um Sistema a Eventos Discretos seja restringido de forma a satisfazer um conjunto de especificações comportamentais.

Ao longo da seção 3.2 foram destacadas as vantagens da abordagem Modular Local para síntese de supervisores em relação à abordagem monolítica. A síntese dos supervisores que coordenam a operação da célula de manufatura introduzida na Seção 3.3 permitiu ilustrar os conceitos apresentados ao longo do capítulo, bem como algumas das vantagens da abordagem Modular Local em relação à abordagem monolítica.

4. Implementação do controle de sistemas a eventos discretos

Apresentam-se, na seção inicial deste capítulo, aspectos tecnológicos e normativos fundamentais referentes à implementação do controle de sistemas a eventos discretos. Inicialmente é abordada a principal solução tecnológica empregada industrialmente na implementação do controle de tais sistemas. Na seqüência são apresentados aspectos relativos à programação de tais equipamentos conforme estabelecidos em norma internacional. Apresentam-se então conceitos elementares para a compreensão da questão da distribuição do controle em múltiplos controladores. Nas subseções seguintes são abordadas duas normas internacionais, a primeira estabelece os princípios de comunicação entre os controladores abordados no início desta seção, a segunda define um modelo de arquitetura para sistemas distribuídos.

A segunda seção deste capítulo apresenta os principais aspectos a serem considerados na implementação do controle de sistemas a eventos discretos quando a síntese dos supervisores é realizada empregando a Teoria de Controle Supervisório. Apresentam-se também os tratamentos para tais aspectos conforme encontrado na literatura.

Finalmente, na última seção, é apresentada uma arquitetura que estrutura o controle de sistemas a eventos discretos com base na aplicação da Teoria de Controle Supervisório. Esta arquitetura realiza o tratamento de parte dos problemas mencionados no parágrafo anterior. O método de implementação a ser apresentado no próximo capítulo é um procedimento sistemático que permite realizar a implementação da referida arquitetura.

4.1 Aspectos tecnológicos e normativos

4.1.1 Controladores Lógico Programáveis

Desenvolvidos no início da década de 1970 para permitir a implementação do controle lógico de sistemas industriais, os Controladores Lógico Programáveis (CLPs) constituem-se atualmente como a principal solução tecnológica adotada no controle dos sistemas produtivos automatizados (HELLGREN *et al.* 2005) (FREY e LITZ, 2000).

Originalmente concebidos para substituir as conexões elétricas entre relés, as quais estabeleciam a programação do controle, os CLPs possuiam arquitetura bastante simples, com capacidade de memória reduzida e um número reduzido de interfaces binárias (entradas e saídas de sinal) com o sistema a ser controlado. Os CLPs atuais são resultantes de uma grande evolução dos equipamentos originalmente desenvolvidos, apresentam elevada capacidade de processamento, grande capacidade de memória, interface amigável com o usuário, interface com o sistema a ser controlado variada e muito abrangente, capacidade de comunicação com outros CLPs e computadores através de protocolos industriais padronizados. Atualmente é possível realizar o diagnóstico de um equipamento e estabelecer ações de controle remotamente, através de uma ligação telefônica com um aparelho celular ou mesmo através de um computador conectado à internet em um *Cyber Caffe*. Outro aspecto que evoluiu bastante ao longo deste tempo diz respeito à programação do CLP. Originalmente os CLPs eram programados através de interfaces gráficas rudimentares que remetiam a diagramas de instalação elétrica. Em geral, a programação era limitada a operações lógicas. Atualmente é possível realizar a programação de CLPs através de linguagens de alto nível, ou de ambientes gráficos amigáveis e com elevada capacidade de estruturação.

Ao longo dos anos foram realizados esforços com o intuito de padronizar a arquitetura e a programação dos CLPs. Na Figura 4.1, adaptada de (FREY e LITZ, 2000), é apresentada a evolução das normas relacionadas aos CLPs. Atualmente a série de normas IEC 61131 - *Programmable Controllers* normatiza diversos aspectos relacionados aos CLPs. Esta série é constituída de sete volumes:

IEC 61131-1 *Programmable controllers - Part* 1: *General information*, 2003, 2nd. ed..

IEC 61131-2 *Programmable controllers - Part* 2: *Equipment requirements and tests*, 2003, 2nd. ed..

IEC 61131-3 *Programmable controllers - Part* 3: *Programming languages*, 2003, 2nd. ed..

IEC/TR 61131-4 *Programmable controllers - Part* 4: *User guidelines* 2004, 2nd. ed..

IEC 61131-5 *Programmable controllers - Part* 5: *Communications*, 2000.

IEC 61131-7 *Programmable controllers - Part* 7: *Fuzzy control programming*, 2000.

IEC/TR 61131-8 *Programmable controllers - Part* 8: *Guidelines for the application and implementation of programming languages*, 2003, 2nd. ed.

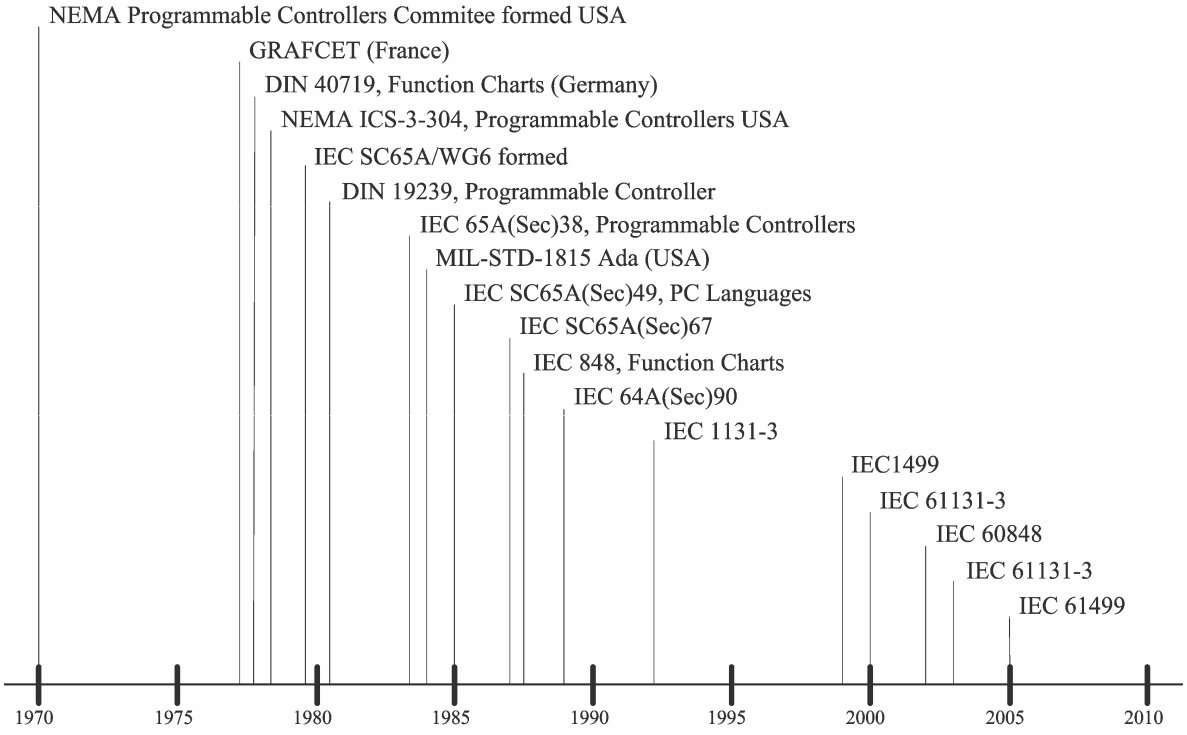


Figura 4.1 – Normatização da programação de CLPs

De acordo com as informações divulgadas pela PlcOpen (PLCOPEN, 2007), uma associação internacional com membros da academia e de diversos setores industriais, o objetivo da IEC 61131-3 é harmonizar a forma como as pessoas projetam e operam controladores industriais. Esta associação é responsável pela certificação da conformidade da interface de programação com a norma. O ideal da associação é atingir um cenário no qual o código desenvolvido para um determinado CLP possa ser implementado diretamente em outro CLP qualquer. Apesar da grande aceitação da IEC 61131-3 por parte dos fabricantes de CLPs, este parece ser um cenário que não será alcançado na prática. Esta associação divulga o grau de conformidade à IEC 61131-3 certificado a diferentes produtos. Por outro lado, a IEC 61131-5, a qual trata da comunicação entre CLPs, não teve uma aceitação satisfatória pela indústria, ao invés disto, cada fabricante de CLP optou em disponibilizar seus próprios blocos de comunicação (PETIG, 2000).

4.1.2 A norma internacional IEC 61131-3

O volume 3 da norma internacional IEC 61131 (IEC, 2003) estabelece os princípios de programação de CLPs. Desde a especificação dos tipos de dados, declaração e representação de variáveis, passando pelo modelo de software até as linguagens de programação (*Ladder Diagram, Function Block Diagram, Instruction List, Structured Text, Sequencial Function Chart*). Esta seção apresenta um breve sumário dos principais conceitos necessários para uma boa compreensão do

método de implementação a ser definido nos próximos capítulos. Uma visão detalhada da norma pode ser encontrada em (JOHN e TIEGELKAMP, 2001) e (LEWIS, 1998).

Um programa de CLP é estruturado empregando três tipos de *Program Organization Units* (POUs). Em ordem crescente de funcionalidade são: *Function*, *Function Block* (FB) e *Program*. Cada POU tem uma seção de código na qual o algoritmo é implementado. Tem, também, uma seção de declaração de variáveis, na qual toda variável empregada na seção de código deve ser declarada. A declaração de uma variável estabelece seu escopo de definição, isto é, a possibilidade de acesso e alteração do conteúdo da referida variável pelo POU correspondente. A norma define os seguintes tipos de variáveis: *var, input, output, in_out, external, global, access path, temp.*

Denomina-se programa de aplicação o conjuntos de POUs desenvolvidos pelo programador do CLP para implementar a lei de controle.

Function Blocks devem ser instanciados no POU que irá invocar a execução de seu algoritmo. Um FB pode ter diversas instâncias. Neste documento, por simplicidade de notação, adotamos a expressão "chamar um FB" como um sinônimo de invocar a execução de uma determinada instância do FB em questão.

A principal diferença entre um *Function* e um *Function Block* é que o último tem a capacidade de reter o valor de variáveis locais entre chamadas consecutivas. Um *Function* não tem esta capacidade. Disto resulta que, quando um *Function* é chamado diversas vezes com os mesmos argumentos (valores das variáveis *input* e *in_out*) ele produz exatamente o mesmos valores em *function result* e nas variáveis *output* e *in_out*.

De acordo com (JOHN e TIEGELKAMP, 2001), um *Sequential Function Chart* (SFC) é uma forma de estruturar a seção de código de um *Program* ou de um FB, bem como, uma linguagem de programação. Um SFC consiste de passos (*steps*) e transições (*transitions*) interconectados por arcos direcionados (*directed links*). A cada passo é associado um conjunto (possivelmente vazio) de ações. A cada transição é associada uma condição de transição, a qual é uma expressão Booleana. Uma ação pode ser uma variável Booleana ou um algoritmo implementado em qualquer linguagem de programação, inclusive um SFC.

Um SFC pode ser representado simplificadamente através de uma tripla ordenada (X, T, x0), onde: X é o conjunto de passos; $T \subseteq X \times X$ é o conjunto de transições; x0 é o passo inicial. Para toda transição $(x_{pred}, x_{suc}) \in T$, com $x_{pred}, x_{suc} \in X$, há um arco direcionado conectando o passo x_{pred} à transição (x_{pred}, x_{suc}) e outro arco direcionado conectando a transição (x_{pred}, x_{suc}) ao passo $x_{suc.}$ Com relação à transição (x_{pred}, x_{suc}) denomina-se x_{pred} o passo predecessor e x_{suc} o passo sucessor a esta transição. HELLGREN *et al.* (2005) assim como BAUER *et al.* (2004) adotam formas mais detalhadas para representação de SFCs. Destaca-se que esta forma de representação é

não-determinística, pois não especifica a prioridade na avaliação das condições de transição nos casos de seleção de seqüências divergentes, isto é, a ordem na qual duas ou mais transições são avaliadas quando tais transições possuem um passo predecessor em comum e passos sucessores distintos.

Um passo de um SFC ou está ativo ou está inativo. Na situação inicial de um SFC o passo inicial está ativo e todos os demais estão inativos. Uma transição está habilitada se todos os passos predecessores a esta transição estão ativos. A liberação de uma transição (*clearing of a transition*) ocorre quando ela está habilitada e a condição de transição a ela associada assume valor lógico VERDADEIRO. A liberação de uma transição causa a desativação de todos os passos predecessores a esta transição e a ativação de todos os passos sucessores à mesma. Isto é referido como evolução do SFC. Uma conseqüência das regras de evolução de SFCs apresentadas em (LEWIS, 1998) é que um passo de um SFC não pode ser ativado e desativado na mesma chamada do FB cuja seção de código é constituída pelo SFC em questão. Em função da forma como as regras de evolução do SFC são efetivamente implementadas, há ambientes de programação em SFC (um caso é o S7-Graph versão 5.3 da SIEMENS (SIEMENS, 2007)) nos quais não é permitido que o passo sucessor de uma transição seja o próprio passo antecessor da transição em questão.

Aspectos mais detalhados de semântica e sintaxe de SFCs são apresentados em (IEC, 2003). HELLGREN *et al.* (2005), BAUER *et al.* (2004) e OHMAN *et al.* (1998) analisam e discutem ambigüidades da norma na semântica de SFCs. Esta análise e discussão está fora do contexto deste documento.

Cada passo em um SFC deve ter um nome distinto, aqui representado genericamente como "step_name". A cada passo é associada uma variável Booleana (step_name.X) que indica se o passo está ativo (step_name.X = VERDADEIRO) ou inativo (step_name.X = FALSO). O tempo transcorrido desde a última ativação do passo é indicado por uma variável do tipo TIME (step_name.T). Quando o passo é desativado o valor de tempo atribuído a esta variável para de ser incrementado. O valor desta variável é estabelecido em zero sempre que o passo é ativado. Tais variáveis são gerenciadas automaticamente pelo sistema.

Inicializar um SFC significa ativar o passo inicial e desativar todos os demais passos. A norma não especifica como isto é realizado. Visto que o valor associado à variável que indica o estado de ativação do passo não pode ser alterado pelo programa de aplicação, então, um procedimento simples para permitir a inicialização do SFC é adicionar um conjunto de transições específico para este fim. Cada transição neste conjunto tem como passo predecessor um passo do SFC e como passo sucessor o passo inicial do SFC. A condição de transição associada a cada uma destas transições é uma variável Booleana do tipo *input* ao FB cuja seção de código é o SFC em questão. O SFC é inicializado sempre que este FB é chamado com o valor lógico VERDADEIRO

associado a esta variável. Alguns ambientes de programação de CLPs possuem um procedimento próprio de inicializar um SFC, em tal caso não é necessário adicionar este conjunto de transições.

A cada ação deve ser estabelecido um qualificador de ação. Ele indica quando e como a ação tem início e fim. Dentre outros, o qualificador "N" estabelece que a ação é executada enquanto o passo ao qual esta ação está associada está ativo. O qualificador "P" estabelece que a ação é executada apenas uma vez na ativação do passo.

Na representação gráfica de um SFC um passo é representado como um retângulo, o passo inicial como um retângulo com uma linha dupla e uma transição como uma barra horizontal.

Vale notar que, apesar das semelhanças entre GRAFCET, conforme definido em (IEC 2002), e SFC, há diferenças conceituais significativas entre estes dois formalismos. "*GRAFCET of IEC 60848 is used to describe/specify the behaviour of system, as viewed from outside of the system, while the SFC language of IEC 61131-3 is used to describe (part of) the implemented software structure inside of the system.*" (IEC, 2002). Enquanto a norma IEC 60848 define um formalismo para descrição comportamental, independente de qualquer tecnologia de implementação, a norma IEC 61131-3 define uma linguagem de programação específica.

Exemplo 4.1)

O SFC que constitui a seção de código do FB representado na Figura 4.2 pode ser representado como (X, T, x0) tal que X = {x0, x1, x2}, T = T' $\cup$ T" com T' = {(x0,x1), (x1,x2), (x2,x1)} e T" = {(x1,x0), (x2,x0)}. Nenhuma ação é associada ao passo x0. Uma ação é associada ao passo x2. A esta ação está associado o qualificador "N". Esta ação é implementada na linguagem *Structured Text*. Quando este passo é ativado (x2.T = T#0s), às variáveis Booleanas "b0" e "g0evt" é estabelecido o valor lógico VERDADEIRO, a variável do tipo *Unsigned Integer* "rspb0" é decrementada de uma unidade. Quando o FB onde este SFC está implementado é chamado novamente, o valor associado à variável "x2.T" é maior do que zero segundos, desta forma à variável "b0" é estabelecido o valor lógico FALSO.

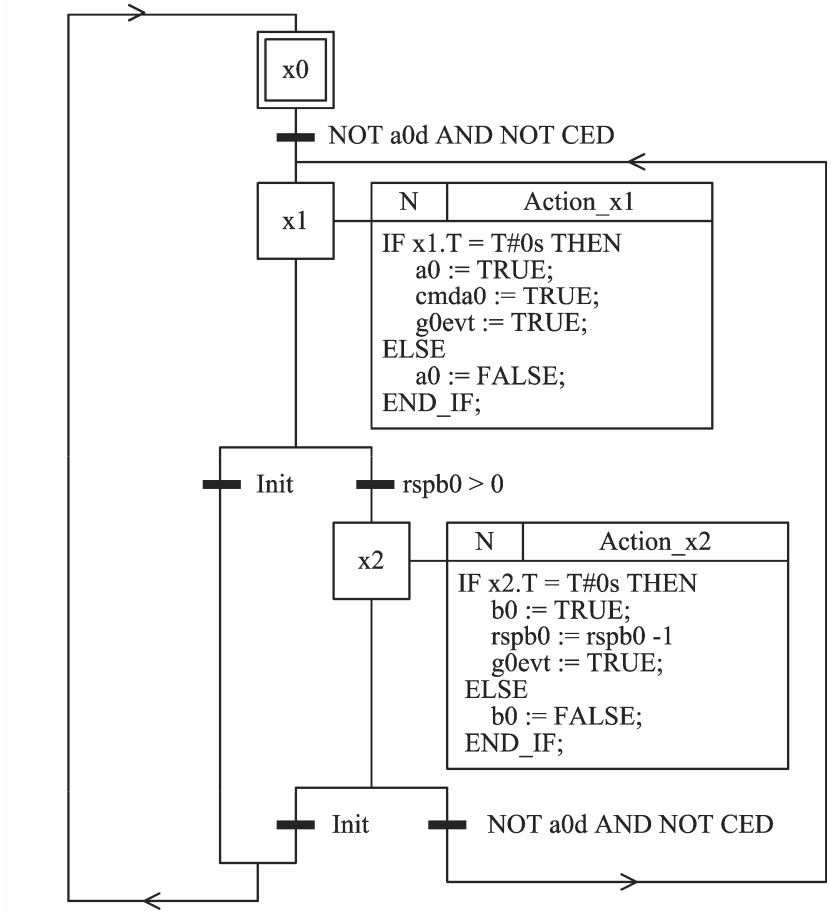
Ainda com relação ao SFC apresentado na Figura 4.2, de forma a ativar o passo x1 é necessário que o passo x0 esteja ativo e que a expressão Booleana associada à transição (x0,x1) tenha valor lógico verdadeiro, ou então, que o passo x2 esteja ativo e que a expressão Booleana associada à transição (x2,x1) tenha valor lógico VERDADEIRO. Neste SFC as duas expressões Booleanas são idênticas, ou seja, as variáveis Booleanas "a0d" e "CED" devem ter o valor lógico FALSO. A condição de transição associada a toda transição em T" é a variável Booleana Init. Esta é uma variável do tipo *input* do FB onde este SFC está implementado. Chamar este FB com o valor lógico VERDADEIRO associado a esta variável causa a inicialização do SFC.

```
Function_Block g0

(* Seção de declaração de variáveis *)

VAR_INPUT
    Init : BOOL;
END_VAR
VAR_EXTERNAL
    a0, a0d, cmda0, b0, g0evt, CED : BOOL;
    rspb0 : UINT;
END_VAR
VAR
    x0.X, x1.X, x2.X : BOOL;
    x0.T, x1.T, x2.T : TIME;
END_VAR

(* Seção de código *)
```




```
END_Function_Block
```

Figura 4.2 – Function Block g0

♦ fim do exemplo 4.1 ♦

4.1.3 Sistemas Distribuídos

"*Interesting applications rarely live in isolation*" (HOHPE e WOOLF, 2004).

Conforme COULOURIS *et al.* (2001), um sistema distribuído é aquele em que componentes em uma rede de computadores se comunicam e coordenam suas atividades apenas através da troca de mensagens.

Conforme CENGIC *et al.* (2006), há um número significativo de padrões aplicáveis a sistemas distribuídos em computadores de uso geral, os mais notáveis são CORBA, DCOM e SOAP. Tecnicamente estes padrões podem ser aplicados a sistemas industriais com controle distribuído, contudo, eles não são apropriados para implementação em CLPs, devido à complexidade para implementação e às limitações de hardware dos CLPs disponíveis atualmente.

Para COULOURIS *et al.* (2001) a motivação para a construção e utilização de sistemas distribuídos está no compartilhamento de recursos. O foco desta Tese é a implementação do controle de sistemas a eventos discretos. Neste sentido, a adoção de uma arquitetura de controle distribuída em diversos controladores é uma conseqüência de um ou mais dos seguintes fatores:

i) a limitação de recursos disponíveis em cada um dos controladores disponíveis para a implementação do controle;

ii) o reaproveitamento de uma estratégia de controle já implementada e validada em um determinado controlador;

iii) a integração de um controlador com características específicas;

iv) a simplificação da reconfiguração do sistema.

Soluções distribuídas precisam transportar dados entre computadores através de uma rede. Este fato acarreta problemas adicionais aos observados em soluções concentradas em um único computador. HOHPE e WOOLF (2004) apontam os principais desafios encontrados na integração de sistemas:

i) as redes de computadores não são confiáveis, é possível que ocorra perda de dados, alteração de conteúdo, duplicação na transmissão, dentre outros problemas;

ii) o transporte de dados através de redes de computadores é diversas ordens de magnitude mais lento do que a execução de um procedimento local;

iii) duas aplicações são inerentemente diferentes (sistema operacional, linguagem de programação, plataforma de operação, formato de dados, ...);

iv) mudanças ao longo do tempo são inevitáveis.

Os referidos autores apontam as principais abordagens adotadas para enfrentar tais desafios:

i) *File Transfer*. Uma aplicação escreve um arquivo que é posteriormente lido por outra aplicação. As aplicações devem concordar em aspectos como o nome e a localização do arquivo, o formato do arquivo, os instante em que ocorre a escrita, a leitura e a exclusão do arquivo;

ii) *Shared Database*. Múltiplas aplicações compartilham uma base de dados comum;

iii) *Remote Procedure Call*. Uma aplicação expõe parte de suas funcionalidades de forma que elas podem ser acessadas remotamente por outras aplicações na forma de um procedimento remoto. Neste caso a comunicação ocorre em tempo real e de forma sincronizada;

iv) *Messaging*. Uma aplicação publica uma mensagem em um canal de comunicação, outras aplicações podem ler o conteúdo da mensagem algum tempo depois. Neste caso a comunicação é, em geral, assíncrona.

HOHPE e WOOLF (2004) comparam as diferentes abordagens, destacando as características positivas e negativas de cada uma delas, porém, o foco destes autores é concentrado na última abordagem. Na seqüência são apresentados alguns conceitos fundamentais sobre esta abordagem.

Messaging é uma tecnologia que permite a comunicação entre programas implementados em computadores distintos. Tal comunicação ocorre em alta velocidade e de forma assíncrona. Os programas se comunicam enviando pacotes estruturados de dados denominados mensagens. Canais de comunicação são caminhos lógicos que conectam os programas e transportam mensagens. *Sender* ou *producer* é um programa que envia uma mensagem, escrevendo-a em um canal de comunicação; *receiver* ou *consumer* é um programa que recebe uma mensagem, lendo-a e excluindo-a do canal de comunicação. O *message system* gerencia o envio e a recepção de mensagens. Esta tecnologia é constituída de cinco etapas:

i) *Create* - o *sender* cria a mensagem e adiciona dados à mesma;

ii) *Send* - o *sender* adiciona a mensagem ao canal de comunicação;

iii) *Deliver* - o *messaging system* transfere a mensagem do computador onde está implementado o *sender* para o computador onde está implementado o *receiver*, disponibilizando-a para este;

iv) *Receive* - o *receiver* lê a mensagem do canal de comunicação;

v) *Process* - o *receiver* extrai dados da mensagem;

Após adicionar a mensagem ao canal de comunicação, o *sender* fica livre para realizar outras atividades enquanto o *message system* realiza a transferência da mensagem. O *sender* pode

confiar que, em algum instante de tempo, o *receiver* irá receber a mensagem e não precisa esperar que isto aconteça.

Os modelos de comunicação abordados em (COULOURIS *et al.*, 2001) enfocam três aspectos fundamentais: interação, falha e segurança. A seguir são apresentados os principais conceitos destes modelos:

Interação:

Os processos interagem através da troca de mensagens, o que resulta na comunicação entre os processos, bem como, na coordenação dos mesmos.

A interação entre processos é influenciada por parâmetros como *latency*, *bandwidth*, *jitter* e *clock*. Os três primeiros parâmetros determinam o desempenho da rede de comunicação. A latência (*latency*) da rede representa o tempo transcorrido entre o início do envio de uma mensagem e o início de sua recepção. A largura de banda (*bandwidth*) representa a quantidade máxima de informações que pode ser transmitido em um determinado intervalo de tempo. Se há diversos computadores trocando informações sobre esta rede, então, eles compartilham este limite. O termo *jitter* representa a variação do tempo necessário para entregar mensagens. Cada computador possui o seu próprio relógio (*clock*). Usualmente não é possível realizar a sincronização de todos estes relógios, além disto, cada relógio apresenta um erro próprio em relação a um relógio ideal. Isto dificulta a adoção de uma noção global de tempo comum a todos os computadores.

Em função dos parâmetros acima referidos os sistemas distribuídos são classificados como síncronos (*synchronous distributed systems*) ou assíncronos (*asynchronous distributed systems*). Estes últimos apresentam as seguintes características: o tempo necessário para executar um determinado processo não apresenta limites pré-estabelecidos; o tempo necessário para transmissão de mensagens é indeterminado; não há sincronização dos relógios dos diversos computadores e o erro dos relógios é indeterminado.

Outro aspecto relevante na interação entre processos diz respeito ao ordenamento com relação ao envio e recepção das mensagens entre os diversos computadores (*event ordering*). Considerando uma base de tempo global, comum a todos os computadores de uma rede, não é assegurado, a priori, que a mensagem M1 (enviada por um determinado computador) seja recebida antes que a mensagem M2 (enviada por um outro computador), mesmo que M1 tenha sido enviada antes de M2.

Falha

Em um sistema distribuído, tanto os computadores como a rede de comunicação são sujeitos a falhas.

Em sistemas assíncronos as falhas são classificadas como falhas por omissão (*omission failures*) e falhas abritrárias (*abritrary* ou *Byzantine failures*).

Uma falha por omissão ocorre quando o processo ou o canal de comunicação deixa de realizar uma ação específica. Por exemplo, quando o processamento de um computador é interrompido. Este tipo de falha pode ser detectado pelos demais computadores do sistema avaliando o tempo transcorrido entre o envio de uma mensagem que requer uma resposta e a chegada desta resposta. Entretanto, em sistemas assíncronos, o transcurso deste tempo não é um indicativo preciso de que houve falha no destinatário da mensagem, o computador pode estar apenas lento, ou executando outras atividades, ou mesmo, porque a mensagem não foi entregue com sucesso. Outro exemplo de falha por omissão é quando uma determinada mensagem não é entregue ao seu destinatário.

Uma falha arbitrária representa o pior tipo de falha que pode acontecer. Por exemplo, quando um processo estabelece um valor incorreto para uma mensagem, ou quando o conteúdo de uma mensagem é adulterado durante a transmissão, ou quando uma mensagem íntegra é entregue repetidas vezes.

Duas propriedades são empregadas para definir o termo comunicação confiável (*reliable communication*):

i) validade - toda mensagem enviada é entregue;

ii) integridade - toda mensagem recebida é idêntica à mensagem enviada e nenhuma mensagem é entregue de forma repetida.

Mecanismos de validação do conteúdo de mensagens (*checksum*) e de verificação de seqüenciamento para identificar a perda ou repetição de mensagens convertem falhas arbitrárias em falhas por omissão. A adoção de tais mecanismos torna confiável a comunicação entre processos.

Segurança

A natureza modular dos sistemas distribuídos os expõe a ataques de agentes externos e mesmo internos. Aspectos como, acesso não autorizado a processos e mensagens, adulteração do conteúdo de mensagens e inserção de mensagens inexistentes afetam negativamente a segurança do sistema. Autenticação de acesso e criptografia de mensagens são mecanismos que podem ser adotados para aumentar a segurança do sistema.

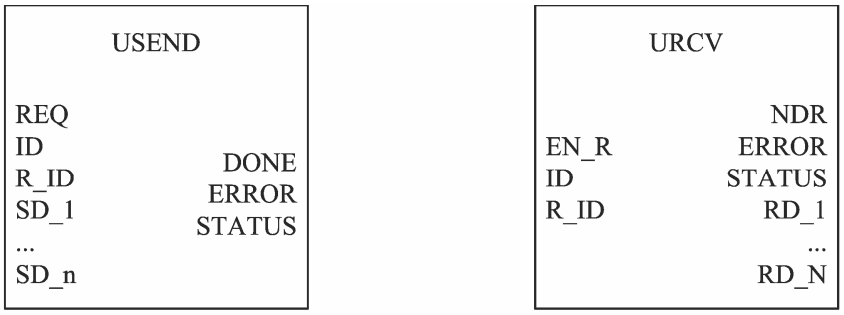
4.1.4 A norma internacional IEC 61131-5

O quinto volume da série de normas IEC 61131 (IEC, 2000) especifica os aspectos da comunicação entre CLPs. Isto é realizado independentemente do sistema de comunicação efetivamente empregado. De acordo com esta norma, a comunicação entre CLPs é realizada através de *Communication Function Blocks*, os quais atuam como uma interface entre o programa de aplicação e o sistema de comunicação.

Dentre outros serviços de comunicação, esta norma define o serviço denominado *Programmed data acquisition*, o qual permite a transmissão de dados entre CLPs. No CLP que produz os dados deve ser implementada uma instância do FB USEND e no CLP que consome os dados deve ser implementada uma instância do FB URCV. Tais FBs possuem como argumentos de entrada a identificação do canal de comunicação (ID) e a identificação do parceiro de comunicação (R_ID). A correta especificação destes argumentos estabelece o canal de comunicação empregado, bem como o ponto de origem e o ponto de destino deste canal. A Figura 4.3 reproduz a interface destes FBs.

Quando requisitado (REQ = VERDADEIRO) a instância do FB USEND toma os dados disponíveis em SD_i, $i \in \{1, ..., n\}$, e envia para a instância correspondente do FB URCV. Dados previamente recebidos são sobre-escritos. Se habilitado, a instância do FB URCV passa os dados recebidos para RD_i, $i \in \{1, ..., n\}$.

Na Figura 4.4 é apresentado o diagrama de transição de estados dos FBs USEND e URCV. As Tabelas 4.1 a 4.4 descrevem as condições que estabelecem as transições de estados, bem como as ações executadas em cada estado.




```
Function_Block USEND

VAR_INPUT
   REQ : BOOL;                  (* Request to send *)
   ID : COMM_CHANNEL;           (* Communication channel *)
   R_ID : STRING;               (* Remote function block *)
   SD_1 : ANY;                  (* User data to send *)
   ...
   SD_n : ANY;                  (* extensible and any type *)
END_VAR

VAR_OUTPUT
   DONE : BOOL;                 (* Function performed *)
   ERROR : BOOL;                (* New non-zero STATUS received *)
   STATUS : INT;                (* Last detected status *)
END_VAR


Function_Block URCV

VAR_INPUT
   EN_R : BOOL;                 (* Enable to receive data *)
   ID : COMM_CHANNEL;           (* Communication channel *)
   R_ID : STRING;               (* Remote function block *)
END_VAR

VAR_OUTPUT
   NDR : BOOL;                  (* New user data received *)
   ERROR : BOOL;                (* New non-zero STATUS received *)
   STATUS : INT;                (* Last detected status *)
END_VAR

VAR_IN_OUT
   RD_1 : ANY;                  (* Received user data *)
   ...
   RD_n : ANY;                  (* extensible and any type *)
END_VAR
```

Figura 4.3 – Interface dos FBs USEND e URCV

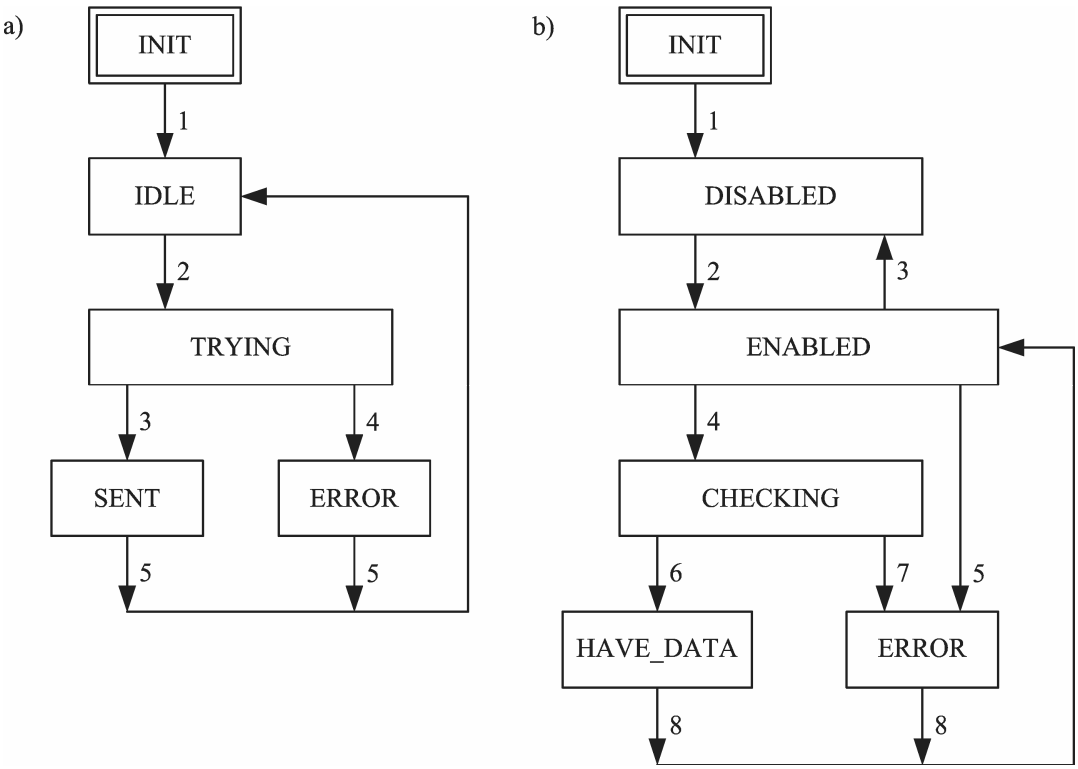


Figura 4.4 – Diagrama de transição de estados: (a) FB USEND; (b) FB URCV

Tabela 4.1 – Transições no FB USEND	
1	Inicialização realizada
2	Na transição positiva de sinal de REQ
3	Sistema de comunicação indica "*Sent to remote communication partner*"
4	Sistema de comunicação indica "Cannot send to remote communication partner" ou outros problemas detectados
5	Após a próxima chamada desta instância do FB

Tabela 4.2 – Transições no FB URCV	
1	Inicialização realizada
2	EN_R = TRUE
3	EN_R = FALSE
4	Dados recebidos do parceiro de comunicação
5	Detectados problemas na comunicação
6	Confirmada compatibilidade do tipo de dados de SD_i e RD_i correspondentes, $i \in \{1, ..., n\}$
7	Identificada alguma incompatibilidade do tipo de dados de SD_i e RD_i correspondentes, $i \in \{1, ..., n\}$
8	Após a próxima chamada desta instância do FB

Tabela 4.3 – Ações executadas no FB USEND				
estado	ação	DONE	ERROR	STATUS
INIT	Inicializa variáveis de saída	FALSE	FALSE	0
IDLE	Sem ações	FALSE	FALSE	-
TRYING	Envia dados em SD_i, $i \in \{1, ..., n\}$, para parceiro de comunicação remoto	-	-	-
SENT	Elimina indicação de erro	TRUE	FALSE	0
ERROR	Indica erro	FALSE	TRUE	*
* o código de erro é atribuído a STATUS				

Tabela 4.4 – Ações executadas no FB URCV					
estado	ação	NDR	ERROR	STATUS	RD_I
INIT	Inicializa variáveis de saída	FALSE	FALSE	0	-
DISABLED	Sem ações	FALSE	FALSE	-	-
ENABLED	Sem ações	-	-	-	-
CHECKING	Verifica compatibilidade de dados	-	-	-	-
HAVE_DATA	Deposita dados em RD_i, $i \in \{1, ..., n\}$	TRUE	FALSE	0	#
ERROR	Indica erro	FALSE	TRUE	*	-
* o código de erro é atribuído a STATUS # os dados recebidos são atribuídos a RD_i, $i \in \{1, ..., n\}$					

4.1.5 A norma internacional IEC 61499

Conforme HUSSAIN e FREY (2005), a introdução da série de normas IEC 61499 marca o início de uma nova era no campo de engenharia de software na área denominada *Industrial Process Measurements and Control System*s (IPMCSs). O objetivo principal desta norma não é o de servir como uma metodologia de programação, mas sim como um modelo de arquitetura para sistemas distribuídos. Até o momento, a maior parte das aplicações desenvolvidas para IPMCSs são monolíticas, difíceis de reconfigurar e modificar. Isto se deve ao fato de que os controladores utilizados são predominantemente CLPs, porém, há a tendência de que outros tipos de controladores passem a ser empregados. O fator motivador para o desenvolvimento desta norma é a inadequação das atuais técnicas para desenvolvimento de software para sistemas distribuídos complexos, que garantam flexibilidade e reconfigurabilidade ao sistema.

FREY e HUSSAIN (2006) destacam que, apesar das similaridades com a IEC 61131 no tocante à estrutura hierárquica e à estrutura atômica dos elementos denominados *Function Blocks*, a IEC 61499 introduz um conceito significativamente inovador. Enquanto esta última introduz um modelo de interação entre *Function Blocks* baseado na ocorrência de eventos (*event driven approach*) a IEC 61131 assume um modelo de execução cíclica (*scan cycle*) onde a comunicação entre *Function Blocks* é baseada na troca de dados ou sinais.

Nos últimos anos tem-se verificado uma intensa atividade do meio acadêmico na investigação de aplicações da IEC 61499. Dentre os autores consultados destacam-se CHRISTENSEN (2000a, 2000b), THRAMBOULIDIS (2005), VYATKIN (2006), FREY e HUSSAIN (2006) e LEWIS (2001).

Apesar do exposto nos parágrafos anteriores, têm-se ressalvas quanto à aplicação industrial imediata dos princípios estabelecidos na IEC 61499. A principal delas diz respeito ao modelo de execução previsto pela IEC 61499, o qual é baseado na ocorrência de eventos que disparam o processamento de algoritmos de controle, e o princípio de processamento dos CLPs disponíveis atualmente, o qual é baseado em uma atualização cíclica do programa. LASTRA e LOBOV (2005) argumentam que os CLPs constituem uma grande parte dos controladores utilizados atualmente na indústria e que, face a grande evolução tecnológica ocorrida ao longo dos anos, aliado à adequação às necessidades dos ambientes industriais, tais controladores não devem ser substituídos em um futuro próximo. Além disto, verifica-se que existe atualmente um número muito reduzido de controladores compatíveis com a IEC 61499.

LASTRA e LOBOV (2005) desenvolvem um aplicativo para um controlador industrial que apresenta características comuns aos CLPs tradicionais e que incorpora características tipicamente encontradas em computadores de uso geral. Este aplicativo permite executar um *Function Block*

Diagram baseado na ocorrência de eventos em um dispositivo que opera segundo um processamento cíclico.

Outra limitação significativa da IEC 61499 diz respeito a ambigüidades na definição do modelo de execução. FERRARINI e VEBER (2004) analisam sete implementações distintas do modelo de execução, todas compatíveis com a norma, e que resultam em comportamentos distintos. CENGIC *et al.* (2006) concluem que um determinado modelo quando implementado em dois sistemas compatíveis com a IEC 61499 apresenta comportamento significativamente distinto.

SUNDER *et al.* (2006) analisam e discutem de forma detalhada ambigüidades na IEC 61499 bem como a aplicabilidade desta norma. Propõem ainda soluções para tais questões. De acordo com estes autores "*widespread adoption of this technology by industry is just in its infancy*."

4.2 Aspectos relativos à Teoria de Controle Supervisório

Conforme HELLGREN *et al.* (2005), apesar da grande aceitação da Teoria de Controle Supervisório pelo meio acadêmico, são raras as aplicações industriais. Uma das principais razões para isto é o problema da implementação física. Tais autores afirmam que há poucas referências de como implementar os supervisores obtidos através da aplicação desta teoria. No caso da implementação em CLPs "*the gap between the event-based automata world and the signal-based PLC-world has to be bridged*".

Ao longo dos últimos anos foram propostas abordagens que buscam resolver o problema apresentado no parágrafo anterior. Dentre as abordagens que tratam da implementação do controle de sistemas a eventos discretos com base na aplicação da Teoria de Controle Supervisório destacam-se: (BALEMI, 1992a, 1992b), (BALEMI *et al.*, 1993), (BONG-SUK e KWANG-HIYUN, 2001), (BRANDIN, 1996), (BRANDIN e CHARBONNIER, 1994), (CENGIC *et al.*, 2005), (CHARBONNIER *et al.*, 1999), (CÔTÉ *et al.*, 2005), (CURZEL e LEAL, 2006), (DIETRICH *et al.*, 2002), (FABIAN e HELLGREN, 1998), (GASPER e MATKO, 2002), (GASPER *et al.*, 2005), (HASDENIR *et al.*, 2004), (HELLGREN, 2000, 2002), (HELLGREN *et al.* 2002), (LAUZON *et al.*, 1996, 1997), (LEDUC, 1996), (LIU e DARABI, 2002), (MALIK, 2002), (MUSIC e MATKO, 2005), (QUEIROZ, 2004), (QUEIROZ e CURY, 2002b), (RAMIREZ *et al.*, 1999), (RAMIREZ-SERRANO *et al.*, 2000), (SILVA *et al.,* 2007a, 2007b), (SANTOS *et al.,* 2006), (TEIXEIRA *et al.*, 2006), (TORRICO, 1999), (VIEIRA *et al.*, 2004a, 2004b 2006a, 2006b, 2007).

Designa-se Arquitetura de Controle Supervisório a estrutura lógica implementada no sistema de controle juntamente com o sistema a ser controlado, tal que a estratégia de controle é obtida através da aplicação da Teoria de Controle Supervisório.

Alguns dos principais problemas que surgem na implementação de arquiteturas de controle supervisório são abordados por FABIAN e HELLGREN (1998), BALEMI (1992a), DIETRICH *et al.* (2002), MALIK (2002) e PARK e CHO (2006). Tais problemas são denominados em (FABIAN e HELLGREN, 1998): causalidade (*causality*); sinais e eventos (*signals and events*); sincronização inexata (inexact *synchronization*) e escolha (*choice*). Além destes, destaca-se ainda a questão da abstração empregada para representar o comportamento livre do sistema a ser controlado durante a síntese do supervisor.

4.2.1 Causalidade

A Teoria de Controle Supervisório (RAMADGE e WONHAM, 1987) assume que todos os eventos são gerados espontaneamente pela planta. Entretanto, conforme discutido em (BALEMI, 1992a) tal hipótese não é apropriada para a maioria dos sistemas reais. Em (FABIAN e HELLGREN, 1998) causalidade está relacionada com a questão "*who generates what*?", isto é, quem é o responsável pela geração de determinados eventos? O supervisor ou a planta?

A Figura 4.5 ilustra uma arquitetura de controle supervisório em conformidade com a hipótese de que todos os eventos são gerados espontaneamente pela planta. O papel dos supervisores é estabelecer a desabilitação de eventos controláveis.

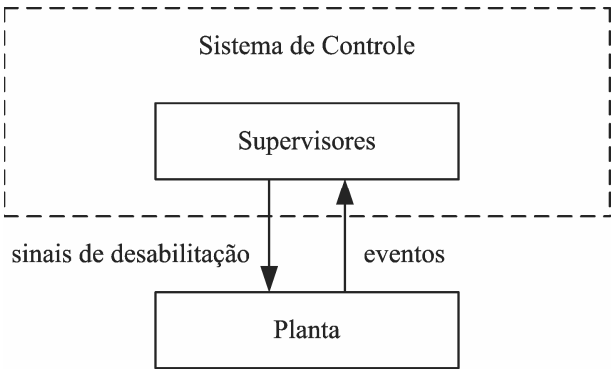


Figura 4.5 – Arquitetura de controle supervisório

GOLASZEWSKI e RAMADGE (1987) e BALEMI (1992a) consideram que uma parcela dos eventos deve ser gerada pelo supervisor. Na abordagem com eventos forçados (GOLASZEWSKI e RAMADGE, 1987) o alfabeto de eventos é particionado como $\Sigma = \Sigma_c \cup \Sigma_{uc} \cup \Sigma_f$. Tanto os eventos controláveis (Σ_c) quanto os eventos não-controláveis (Σ_{uc}) ocorrem espontaneamente no sistema a ser controlado, já os eventos forçados (Σ_f) ocorrem se e somente se são forçados pela ação de controle do supervisor. Esta ação consiste na habilitação ou desabilitação de eventos controláveis e na geração de eventos forçados. Nesta abordagem as condições para existência do supervisor não são alteradas em relação ao estabelecido por

RAMADGE e WONHAM (1987), porém, é realizada a redefinição da controlabilidade de linguagens.

Na abordagem entrada/saída (BALEMI, 1992a) os eventos controláveis são designados comandos (*controller outputs*) e os eventos não-controláveis são designados respostas (*plant outputs*). Nesta abordagem, ao invés de habilitar ou desabilitar eventos controláveis, o supervisor tem a função de gerar comandos. A redefinição do papel do supervisor altera as condições para sua existência. Conforme BALEMI (1992a), as condições de existência de supervisores estabelecidas por RAMADGE e WONHAM (1987) requerem que o supervisor seja completo em relação à planta, isto é, o supervisor deve ser sempre capaz de seguir (mapear) a ocorrência de qualquer evento não-controlável gerado pela planta, não podendo impedir a ocorrência de qualquer evento não-controlável. Na abordagem entrada/saída, além desta condição, é necessário que a planta seja completa em relação ao supervisor, isto é, a planta sempre deve ser capaz de aceitar os eventos controláveis gerados pelo supervisor. Conforme BALEMI (1992a) esta condição é satisfeita sempre que $L(\mathbf{S}) \subseteq L(\mathbf{G})$, onde **G** é o autômato que representa o comportamento livre do sistema a ser controlado e **S** é o autômato que representa o supervisor sintetizado em conformidade com a abordagem de RAMADGE e WONHAM (1987). Destaca-se que, para o caso geral, o autômato que representa um supervisor reduzido não satisfaz tal propriedade.

FABIAN e HELLGREN (1998) apresentam como converter o autômato que representa de forma não-reduzida um supervisor global (que satisfaz ao conjunto de especificações como um todo) em um programa de CLP expresso na linguagem *Ladder Diagram*. No programa de CLP resultante, o supervisor é o responsável pela geração de eventos controláveis enquanto o sistema a ser controlado é o responsável pela geração dos eventos não-controláveis. A aplicação deste método não é direta para o caso em que a síntese de supervisores é realizada empregando uma abordagem modular. Se há um conjunto de supervisores modulares responsáveis pela geração de um determinado evento controlável, então, é necessário que haja um consenso destes supervisores com relação à geração de tal evento. BALEMI (1992a) e LEDUC (1996) realizam o tratamento desta questão, permitindo a implementação de supervisores sintetizados sob uma abordagem modular. A aplicação do método proposto por FABIAN e HELLGREN (1998) também não permite o emprego de representações reduzidas do supervisor.

Na Figura 4.6 é ilustrado o procedimento proposto por FABIAN e HELLGREN (1998) para implementação de supervisores em linguagem *Ladder Diagram*. De acordo com este procedimento, a ocorrência de um evento não-controlável é detectada através da transição positiva de sinal de uma variável Booleana de entrada do CLP (na referida figura, eventos $\beta 1$ e $\beta 2$ e variáveis b1 e b2). A geração de um evento controlável resulta na ativação, durante um ciclo de

atualização do CLP, de uma variável Booleana de saída do CLP (evento α1 e variável a1). A cada estado do supervisor é associada uma variável Booleana (estados e variáveis q0 a q4).

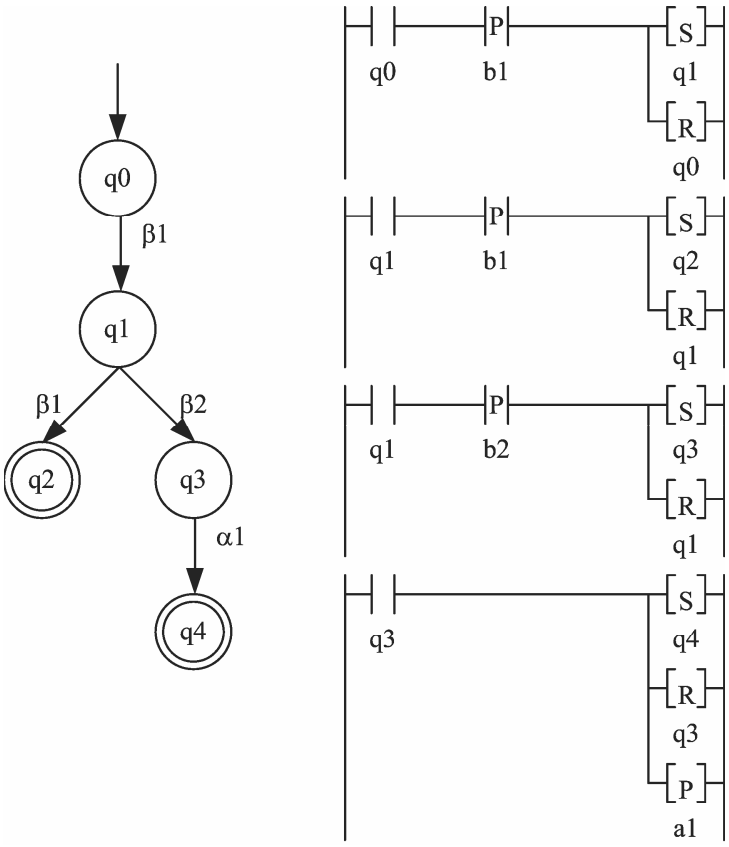


Figura 4.6 – Supervisor e implementação em *Ladder Diagram*

Em CHARBONNIER *et al.* (1999) é proposta uma arquitetura de controle na qual um conjunto de supervisores modulares coordena o comportamento do sistema através da habilitação ou desabilitação de eventos. Os eventos são gerados em um nível intermediário da arquitetura de controle, implementado entre os supervisores e o sistema a ser controlado. Uma forte limitação desta proposta é que os supervisores são obtidos empiricamente, sem a aplicação da Teoria de Controle Supervisório.

4.2.2 Sinais e eventos

FABIAN e HELLGREN (1998) caracterizam esta categoria de problemas através das seguintes considerações. A Teoria de Controle Supervisório considera que os eventos ocorrem assincronamente em instantes de tempo discretos e que a ocorrência de eventos determina a transição do estado ativo dos supervisores. Por outro lado, os CLPs processam variáveis cujos valores são atualizados ciclicamente. Estes autores enquadram nesta categoria o efeito avalanche e a incapacidade de reconhecer a ordem de eventos. Considera-se que a forma para realizar o

tratamento de eventos proposta pelos referidos autores não é apropriada para o caso geral, podendo resultar em outros dois problemas não abordados na literatura, aqui denominados perda de informação e duração dos comandos.

Efeito avalanche

O efeito avalanche é verificado quando a ocorrência de um evento resulta na ativação e desativação da variável associada a um estado do supervisor ao longo de um único ciclo de atualização do CLP. Este fato corresponde a múltiplas transições de estado devidas a uma única ocorrência do evento, o que resulta na quebra da sincronia entre o sistema a ser controlado e o supervisor implementado no CLP. Com isto as próximas ações de controle serão, possivelmente, inválidas. O exemplo 4.2 ilustra a ocorrência do efeito avalanche.

Exemplo 4.2)

Supondo que o estado ativo do autômato apresentado na Figura 4.6 é o estado q0, a ocorrência do evento β1 deve resultar na ativação do estado q1. Contudo, na implementação representada nesta figura a ocorrência deste evento resulta na ativação do estado q2, pois o estado q1 será ativado e desativado ao longo de um único ciclo de atualização do CLP. Ou seja, uma ocorrência do evento β1 terá o efeito de duas ocorrências consecutivas deste evento.

♦ fim do exemplo 4.2 ♦

FABIAN e HELLGREN (1998) propõem que a estrutura lógica associada a cada transição de estado seja implementada em ordem reversa à apresentada na Figura 4.6. Esta solução é eficiente desde que não existam ciclos na estrutura de transição de estados do autômato. Os referidos autores não apresentam qualquer solução factível para o caso em que existam tais ciclos.

HASDEMIR *et al.* (2004) propõem que a cada estado do supervisor sejam relacionadas duas variáveis do CLP. Uma variável é relacionada ao estado lógico do estado do supervisor no ciclo de atualização corrente do CLP e a outra variável é relacionada ao estado lógico deste mesmo estado do supervisor no ciclo de atualização seguinte. Os referidos autores demonstram como realizar a ativação e desativação destas variáveis. Com esta solução, a ordem em que são implementadas as lógicas relacionadas a cada transição de estado é irrelevante, sendo aplicável para autômatos em que existam ciclos na estrutura de transição de estados. Porém, tem como limitação o fato de aumentar o número de variáveis necessárias à implementação do autômato.

Propõe-se um procedimento mais simples para garantir que o efeito avalanche não irá ocorrer. Tal procedimento consiste em associar uma única variável do CLP ao supervisor como um todo. Tal variável é ativada sempre que ocorre uma transição de estado. Esta variável é desativada no final da seção de código do POU onde a estrutura do supervisor está implementada. Toda

transição de estado inclui como condição a negação desta variável. Na Figura 4.7 é apresentada a implementação correspondente ao supervisor apresentado na Figura 4.6.

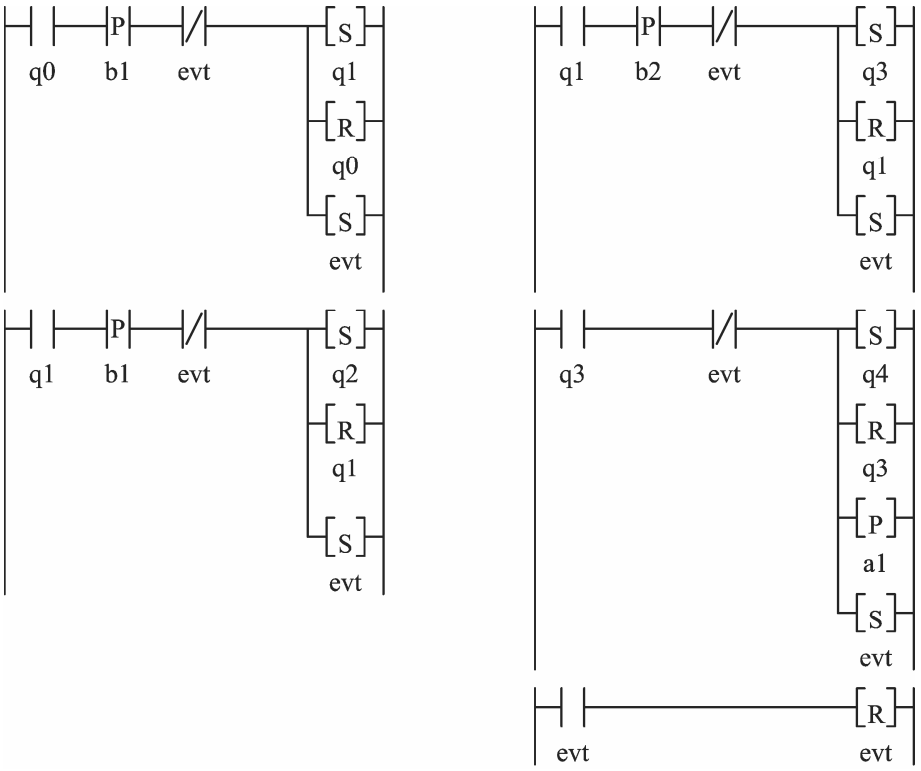


Figura 4.7 – Implementação de supervisor imune ao efeito avalanche

Incapacidade de reconhecer a ordem de eventos

A Teoria de Controle Supervisório assume que dois eventos quaisquer nunca ocorrem simultaneamente. Contudo, é plenamente razoável supor que, ao longo de um determinado ciclo de atualização do CLP, ocorra a transição positiva de sinal de duas variáveis Booleanas de entrada do CLP. Neste caso, não é possível determinar a ordem de ativação de tais variáveis. Se a ocorrência de eventos não-controláveis é identificada através da transição positiva de sinal de tais variáveis, então, não é possível determinar a ordem de ocorrência de tais eventos.

Conforme FABIAN e HELLGREN (1998) um supervisor $\mathfrak{S}$ é insensível ao entrelaçamento (*interleave insensitive*) com relação a uma planta **G** (definida sobre o alfabeto $\Sigma = \Sigma_c \cup \Sigma_{uc}$) e um subalfabeto Σ_c se, para $s \in \Sigma^*$, $s1$, $s2 \in \Sigma_{uc}^*$ e $\sigma \in \Sigma_c$ tem-se que:

$$s\ s1\ s2\ \sigma \in L(\mathfrak{S}/\mathbf{G}) \rightarrow s\ (s1\ |||\ s2)\ \sigma \subseteq L(\mathfrak{S}/\mathbf{G})$$

Ou seja, um supervisor é insensível ao entrelaçamento se sua ação de controle não depende dos possíveis entrelaçamentos de eventos não-controláveis que podem ocorrer após qualquer seqüência s.

Caso esta propriedade não seja satisfeita, o supervisor pode adotar uma ação de controle incorreta em função de não poder reconhecer a ordem de ocorrência dos eventos não-controláveis. Este problema não ocorre se o supervisor é insensível ao entrelaçamento.

Exemplo 4.3)

Na Figura 4.8 é apresentado um supervisor cujo objetivo é distinguir a ordem de ocorrência dos eventos β1 e β2 e habilitar/desabilitar adequadamente os eventos α1 e α2. Após a seqüência β1β2 é habilitado o evento α1 (desabilitado o evento α2) e após a seqüência β2β1 é habilitado o evento α2 (desabilitado o evento α1).

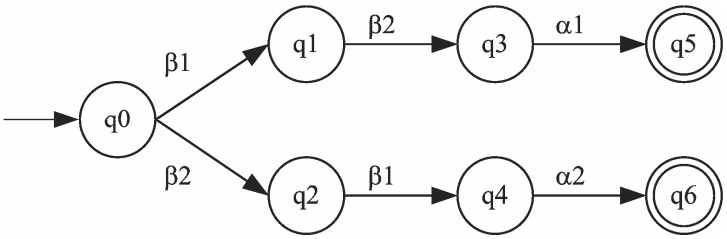


Figura 4.8 – Supervisor para distinção de seqüências de eventos
(FABIAN e HELLGREN, 1998)

Considerando que o autômato apresentado na Figura 4.8 gera a linguagem $L(\mathcal{S}/\mathbf{G})$, e tomando $s = \varepsilon \in \Sigma^*$, $s1 = \beta1 \in \Sigma_{uc}^*$, $s2 = \beta2 \in \Sigma_{uc}^*$, $\sigma = \alpha1 \in \Sigma_c$, tem-se que:

i) $s\ s1\ s2\ \sigma = \beta1\beta2\alpha1 \in L(\mathcal{S}/\mathbf{G})$;

ii) $s\ (s1\ |||\ s2)\ \sigma = \{\ \beta1\beta2\alpha1\ ,\ \beta2\beta1\alpha1\ \}$;

Como $\beta2\beta1\alpha1 \notin L(\mathcal{S}/\mathbf{G})$ então o supervisor não é insensível ao entrelaçamento e sua ação de controle não é confiável.

♦ fim do exemplo 4.3 ♦

Perda de informação

Para evitar o efeito avalanche, a implementação deve ser realizada de forma que cada uma das variáveis associadas aos estados do supervisor não sejam ativadas e desativadas ao longo de um único ciclo de atualização do CLP. Disto resulta que, ao longo de cada ciclo de atualização do CLP, será realizado o tratamento de, no máximo, uma ocorrência de evento.

Se a ocorrência de eventos não-controláveis é identificada através da transição positiva de sinal de variáveis de entrada do CLP, então, sempre que dois eventos não-controláveis ocorrerem ao longo de um determinado ciclo de atualização do CLP o estado ativo do supervisor será

atualizado com a ocorrência de apenas um destes eventos. A ocorrência do outro evento não será processada. Este fato resulta em perda da informação e na quebra da sincronia entre o sistema a ser controlado e o supervisor implementado no CLP.

Exemplo 4.4)

Considere que o estado ativo do supervisor apresentado na Figura 4.6 é o estado q0, e que, ao longo de um determinado ciclo de atualização, ocorram os eventos não-controláveis β1 e β2, resultando na ativação simultânea das variáveis b1 e b2. Considere, também, que para evitar o efeito avalanche a implementação apresentada na Figura 4.6 tenha sido realizada em ordem reversa, ou que tenha sido adotada a implementação apresentada na Figura 4.7. No ciclo de atualização em que ocorre a ativação das referidas variáveis será realizado o tratamento da ocorrência do evento β1, resultando na ativação da variável associada ao estado q1. Porém, a transição de estado devida à ocorrência do evento β2 não será tratada neste ciclo de atualização do CLP. Como nos próximos ciclos de atualização a avaliação da transição positiva de sinal da variável b2 resulta em valor lógico FALSO, a ocorrência do evento β2 não será tratada novamente.

♦ fim do exemplo 4.4 ♦

Duração dos comandos

Com a proposta de FABIAN e HELLGREN (1998) para a geração de eventos pelo supervisor, a duração dos mesmos é limitada a um ciclo de atualização do CLP utilizado na implementação. Verifica-se que a duração deste sinal elétrico pode não ser suficientemente longa para sensibilizar adequadamente os elementos de atuação do sistema ou mesmo para que seja detectada por outro dispositivo de controle. Por outro lado, a duração deste sinal elétrico pode ser excessivamente longa, fazendo com que o elemento de atuação ou outro dispositivo de controle a interprete como correspondente à ocorrência de múltiplos eventos.

Destaca-se ainda que, no caso em que a implementação é realizada em diferentes CLPs, com diferentes tempos de ciclo de atualização, existe a possibilidade que tal fenômeno seja verificado em um CLP mas não seja verificado em outro.

4.2.3 Escolha

A aplicação da Teoria de Controle Supervisório garante que o comportamento obtido sob supervisão é minimamente restritivo. Desta forma, após a ocorrência de uma determinada seqüência de eventos, pode haver múltiplos eventos (controláveis e/ou não-controláveis) habilitados, ou seja, pode haver múltiplas opções para prosseguimento do comportamento gerado sob supervisão. Sob a hipótese de que o sistema a ser controlado gera todos os eventos, então, este

sistema define qual o próximo evento a ser gerado. Porém, quando o sistema de controle é responsável pela geração de uma parcela de eventos, então, a decisão sobre qual o próximo evento a ser gerado é compartilhada entre o sistema a ser controlado e o sistema de controle.

Do exposto em (FABIAN e HELLGREN, 1998) conclui-se que, quando há múltiplos eventos controláveis habilitados, apenas um deve ser gerado até que ocorra a atualização de estado dos supervisores. Isto é necessário, pois a geração de um evento controlável pode estabelecer a desabilitação de eventos que estavam habilitados. Além disto, a implementação deve priorizar o tratamento dos eventos não-controláveis em detrimento dos eventos controláveis. Em geral, a forma como é realizada a implementação define a ordem de tratamento de eventos.

Conforme MALIK (2002), ao priorizar a geração de um determinado evento controlável em detrimento dos demais, uma parcela do comportamento obtido sob ação de supervisão deixa de ser realizável. É possível que tal escolha resulte no bloqueio do comportamento do sistema sob ação de supervisão.

Exemplo 4.5)

Considere o sistema constituído por **G1** e **G2** cujo comportamento é representado na Figura 4.9, onde αi e βi, com i = {1,2}, correspondem, respectivamente, ao início e fim de operação dos equipamentos **G1** e **G2**. Como especificação de controle, representada por **E**, deseja-se evitar que os dois equipamentos estejam em operação simultaneamente e que sempre seja possível operar o equipamento **G2**.

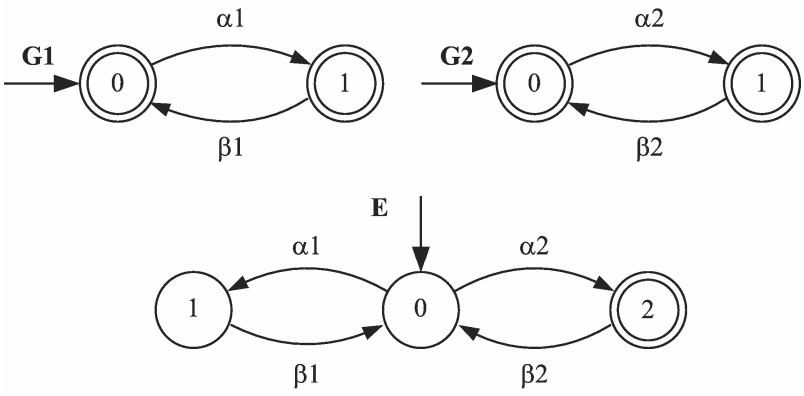


Figura 4.9 – Sistema a ser controlado e especificação de controle

Pode-se verificar que o comportamento desejado (L_m(**E**)) é não-bloqueante e controlável em relação a L(**G**), onde **G** = **G1** || **G2**. Portanto existe um supervisor $\mathcal{S}$ tal que $L_m(\mathcal{S}/\boldsymbol{G}) = L_m(\mathbf{E})$. Caso o evento $\alpha 1$ tenha prioridade de geração em relação ao evento $\alpha 2$, então, este último nunca

será gerado e o comportamento efetivamente obtido sob supervisão será bloqueante, conforme apresentado na Figura 4.10.

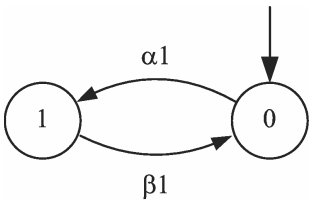


Figura 4.10 – Comportamento efetivamente obtido sob supervisão

Caso o evento α2 tenha prioridade de geração em relação ao evento α1, então, este último nunca será gerado. O problema do bloqueio não é mais observado, mas persiste o problema de que o comportamento efetivamente obtido sob supervisão corresponde a apenas uma parcela do comportamento previsto. Neste caso, o equipamento **G1** nunca entrará em operação.

♦ fim do exemplo 4.5 ♦

DIETRICH *et al.* (2002) definem um conjunto de propriedades que, se simultaneamente satisfeitas, garantem que qualquer implementação realizada para o supervisor produz um comportamento não-bloqueante.

4.2.4 Sincronização inexata

Conforme PARK e CHO (2006), a abordagem em (RAMADGE e WONHAM, 1987) assume que a ocorrência de qualquer evento não-controlável no sistema a ser controlado é imediatamente observada pelo supervisor. Assume, também, que a ação de controle estabelecida pelo supervisor é aplicada imediatamente ao sistema a ser controlado. Contudo, tais hipóteses não são realísticas para o caso geral, pois a comunicação entre o sistema de controle onde está implementado o supervisor e o sistema a ser controlado é sujeita a atrasos significativos.

Quando a observação da ocorrência de um evento não-controlável resulta na alteração da ação de controle, é possível que, durante o tempo necessário para que o supervisor seja notificado da ocorrência do evento, a ação de controle, já inválida, seja aplicada indevidamente ao sistema a ser controlado. Este problema é denominado sincronização inexata (*inexact synchronization)* por FABIAN e HELLGREN (1998).

Exemplo 4.6)

O supervisor realizado pelo autômato apresentado na Figura 4.11 está sujeito à sincronização inexata. Sendo q1 o estado ativo do supervisor, a ação de controle consiste na habilitação (geração) do evento controlável α. Durante o tempo necessário para avaliar o programa

de controle e efetivar a ação de controle, é possível que ocorra o evento não-controlável β2, o que requer a desabilitação do evento α, porém, a ocorrência de tal evento não é notificada imediatamente ao supervisor. Desta forma, o evento α terá sido gerado indevidamente pelo sistema de controle, além disto, o supervisor não poderá mapear a ocorrência do evento β2, pois o estado ativo do mesmo (estado q3) não prevê a ocorrência deste evento. Este fato resulta na quebra da sincronia entre o sistema a ser controlado e o supervisor implementado no CLP.

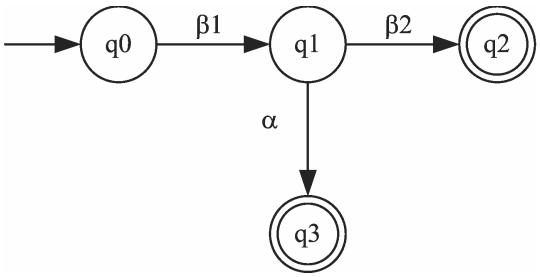


Figura 4.11 – Supervisor sujeito ao fenômeno da sincronização inexata
(FABIAN e HELLGREN, 1998)

♦ fim do exemplo 4.6 ♦

MALIK (2002) define uma propriedade que, se satisfeita pelo modelo que descreve o comportamento livre do sistema a ser controlado garante que o problema da sincronização inexata não irá ocorrer. BALEMI (1992a) e PARK e CHO (2006) abordam este problema de forma diferente. Caso as propriedades definidas pelos referidos autores sejam satisfeitas pela linguagem que representa o comportamento sob supervisão, então, está garantido que a sincronização inexata não ocorre.

Caso a planta satisfaça a propriedade definida por MALIK (2002), então, é possível realizar a síntese e implementação dos supervisores sem precisar considerar os aspectos da comunicação entre o sistema a ser controlado e o sistema de controle. Caso não seja possível representar o comportamento livre do sistema a ser controlado através de um modelo que satisfaça tal propriedade, é necessário verificar se o comportamento obtido sob supervisão satisfaz as propriedades definidas por BALEMI (1992a) ou por PARK e CHO (2006).

Seja **G** = (Q, Σ, δ, q0, Q_m) um autômato determinístico. O autômato **G** é Σ_c-Σ_{uc}-comutador (Σ_c-Σ_{uc}-*commuting*) se para todo $q \in Q$, $\sigma_c \in \Sigma_c$ e $\sigma_{uc} \in \Sigma_{uc}$ tal que $\delta(q,\sigma_c)!$ e $\delta(q,\sigma_{uc})!$ tem-se que $\delta(q,\sigma_c\sigma_{uc})!$ bem como que $\delta(q,\sigma_{uc}\sigma_c)!$ e que $\delta(q,\sigma_c\sigma_{uc}) = \delta(q,\sigma_{uc}\sigma_c)$ (MALIK, 2002).

Conforme MALIK (2002), se a planta é Σ_c-Σ_{uc}-comutadora, então as seguintes propriedades são satisfeitas:

1) A planta aceita qualquer mensagem (evento) do controlador, mesmo que esta mensagem esteja atrasada;

2) Cada controlador, sintetizado a partir de um modelo contendo a planta, aceita qualquer mensagem da planta, mesmo que esta mensagem esteja atrasada;

3) Se toda mensagem é recebida, ou seja, nenhuma mensagem é perdida na comunicação entre a planta e o controlador, então o controlador pode estabelecer o estado da planta.

Exemplo 4.7)

Na Figura 4.12 é apresentado um modelo para representação do comportamento livre de um sistema. No autômato **G** tem-se que $\Sigma_c = \{\alpha, \gamma\}$ e $\Sigma_{uc} = \{\beta\}$, onde α representa início de operação, γ representa interromper a operação e β representa fim de operação. Tem-se que $\delta(1, \gamma)!$ e $\delta(1,\beta)!$, porém, $\delta(1, \gamma\beta)$ não está definido, nem tão pouco $\delta(1, \beta\gamma)$. Assim, o autômato **G** não é Σ_c-Σ_{uc}-comutador.

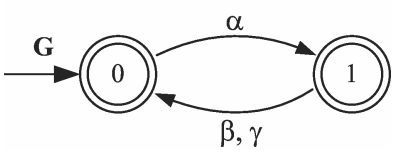


Figura 4.12 – Sistema não-Σ_c-Σ_{uc}-comutador (MALIK, 2002)

♦ fim do exemplo 4.7 ♦

Conforme BALEMI (1992a) uma linguagem *K* é insensível ao atraso (*delay insensitive language*) se para $s \in \overline{K}$, $\sigma_c \in \Sigma_c$, $\sigma_u \in \Sigma_{uc}$ tem-se que $(s\sigma_c,\ s\sigma_u \in \overline{K}) \rightarrow s\sigma_u\sigma_c, s\sigma_c\sigma_u \in \overline{K}$. Se a linguagem obtida sob supervisão ($L_m(\mathcal{S}/\mathbf{G})$) é insensível ao atraso, então, o problema da sincronização inexata não ocorre.

Exemplo 4.8)

Considere que o autômato apresentado na Figura 4.11 representa a linguagem obtida sob supervisão, e que $\Sigma_c = \{\alpha\}$, $\Sigma_{uc} = \{\beta1, \beta2\}$.

Sendo $\overline{K} = \{\varepsilon, \beta1, \beta1\beta2, \beta1\alpha\}$ e tomando $s = \beta1$, $\sigma_c = \alpha$ e $\sigma_u = \beta2$, tem-se que $(s\sigma_c = \beta1\alpha \in \overline{K})$ e $(s\sigma_u = \beta1\beta2 \in \overline{K})$, porém, $(s\sigma_u\sigma_c = \beta1\beta2\alpha \notin \overline{K})$ nem tão pouco $(s\sigma_c\sigma_u = \beta1\alpha\beta2 \notin \overline{K})$. Disto conclui-se que a linguagem $K = \{\beta1\beta2, \beta1\alpha\}$ não é insensível ao atraso.

♦ fim do exemplo 4.8 ♦

PARK e CHO (2006) definem as condições para as quais uma dada linguagem é *Delay-nonconflicting* em relação a um autômato e um número D de ocorrências de eventos não-controláveis. Os autores demonstram formalmente que o problema discutido nesta seção não é observado se e somente se $L_m(\mathfrak{K}/\mathbf{G})$ é delay-nonconflicting em relação à **G** e D. Os referidos autores apresentam ainda um algoritmo que permite verificar se esta propriedade é satisfeita.

FABIAN e HELLGREN (1998) destacam que os problemas denominados sincronização inexata (*inexact synchronization*) e insensibilidade ao entrelaçamento (*interleave insensitivity*) são distintos entre si. Enquanto este último está relacionado à incapacidade de distinguir a ordem de ocorrência de dois ou mais eventos gerados no sistema a ser controlado, o primeiro está relacionado a incapacidade de observar a ocorrência de novos eventos enquanto a decisão de controle é estabelecida.

4.2.5 Detalhamento da abstração

Um importante aspecto não abordado na literatura refere-se à abstração aplicada na modelagem do comportamento do sistema a ser controlado. Geralmente o modelo considera apenas parte dos eventos efetivamente relacionados ao comportamento do sistema. Ele abstrai as atividades que devem ser realizadas quando da ocorrência de eventos, bem como os sinais trocados entre o sistema de controle e o sistema a ser controlado. Tal abstração é importante para evitar a explosão do número de estados do modelo e do supervisor durante a síntese do mesmo. Geralmente são considerados apenas os eventos necessários para coordenar o comportamento concorrente dos diversos subsistemas.

4.3 Arquitetura de controle supervisório segundo Queiroz e Cury

A arquitetura de controle supervisório proposta por QUEIROZ e CURY (2002b) permite o adequado tratamento de parte das questões apresentadas na seção anterior. Isto é realizado através da introdução de uma interface entre os supervisores sintetizados em conformidade com a Teoria de Controle Supervisório e o sistema a ser controlado, esta interface é constituída pelo nível Sistema Produto e o nível Procedimentos Operacionais (Figura 4.13).

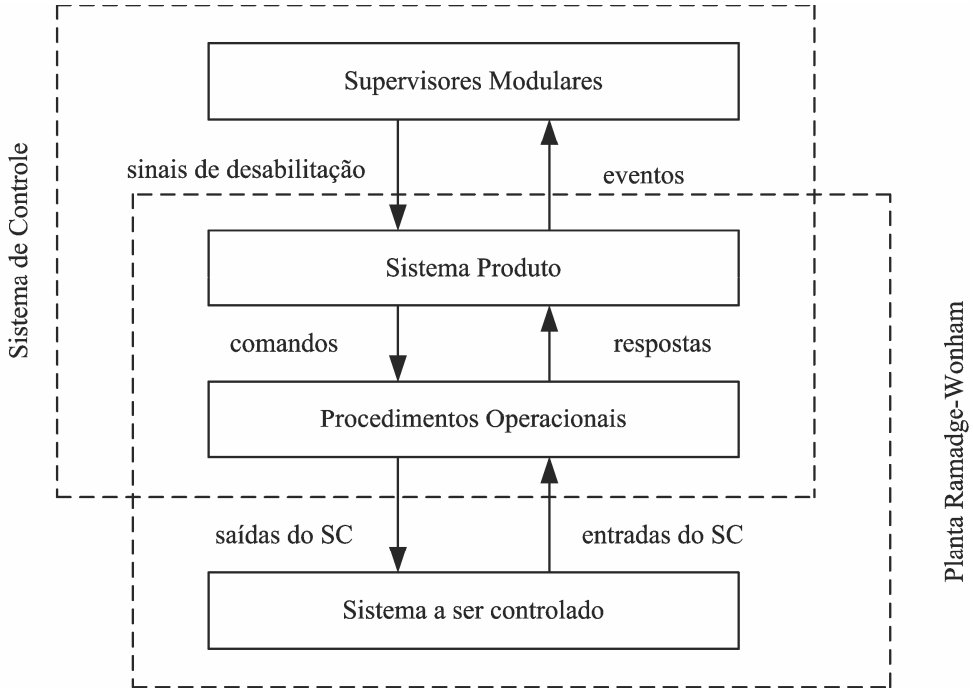


Figura 4.13 – Arquitetura de controle supervisório segundo QUEIROZ e CURY

Esta arquitetura de controle é estruturada da seguinte forma:

i) O nível Supervisores Modulares (SM) é constituído pelo conjunto de supervisores obtidos com a aplicação da abordagem Modular Local da Teoria de Controle Supervisório ($\{\mathfrak{S}_j \mid j \in J\}$), ou num caso particular, um único supervisor global;

ii) O nível Sistema Produto (SP) é constituído de um conjunto de estruturas denominadas "módulos do sistema produto" ($\{\mathbf{g_i} \mid i \in I\}$), cada uma delas associada a um subsistema da representação por sistema produto $\{\mathbf{G_i} \mid i \in I\}$ empregada na representação do comportamento livre do sistema a ser controlado;

iii) O nível Procedimentos Operacionais (PO) é constituído por um conjunto de estruturas denominadas "procedimento operacionais" ($\{o_\sigma, \sigma \in \Sigma_c\}$). Geralmente, a cada evento controlável empregado na representação do comportamento livre do sistema a ser controlado é associado um procedimento operacional;

iv) O sistema a ser controlado.

A comparação desta arquitetura de controle com a apresentada na Figura 4.5 demonstra que a planta é constituída pelos níveis Sistema Produto, Procedimentos Operacionais e Sistema a ser controlado.

Em geral, a modelagem do sistema considera apenas os eventos necessários para expressar a coordenação dos subsistemas. A associação de procedimentos operacionais aos eventos

empregados na representação do comportamento do sistema permite realizar o detalhamento das abstrações realizadas na etapa de modelagem. Desta forma, ações de controle que podem ser estabelecidas de forma trivial podem ser deixadas a cargo da modelagem dos procedimentos operacionais e abstraídas durante a modelagem do sistema e das especificações de controle. Tal prática resulta em modelos mais simples para descrição do comportamento independente dos subsistemas (modelos com menor número de estados e/ou eventos) e evita a formulação de especificações de controle. Isto reduz a complexidade do processo de síntese dos supervisores.

A comunicação entre os diferentes níveis desta arquitetura de controle ocorre empregando eventos, sinais de desabilitação, comandos, respostas e sinais de entrada e saída do sistema de controle. A cada evento controlável é associado um comando e um sinal de desabilitação. A cada evento não-controlável é associada uma resposta. Os sinais de entrada e saída do sistema de controle são estabelecidos com base nos atuadores e sensores que efetivamente constituem o sistema a ser controlado.

O módulo do sistema produto associado ao subsistema $\mathbf{G_i}$, $i \in I$, implementa o comportamento independente de $\mathbf{G_i}$. Esta estrutura é a responsável pela geração de eventos controláveis em Σ^{Gi}. Ela também sinaliza a ocorrência de eventos não-controláveis em Σ^{Gi}. A geração de um evento controlável resulta na ativação do comando associado ao evento em questão. Os eventos tratados (eventos controláveis gerados ou a sinalização da ocorrência dos eventos não-controláveis) são enviados para o nível Supervisores Modulares, os comandos ativos são enviados para o nível Procedimentos Operacionais.

Quando um procedimento operacional identifica a ativação de um comando, ele dispara a execução das atividades associadas à ocorrência do evento controlável associado. A execução de tais atividades pode resultar na ocorrência de eventos não-controláveis. Quando um procedimento operacional identifica a ocorrência de um evento não-controlável ele ativa a resposta associada ao evento em questão. Esta resposta é enviada ao nível Sistema Produto, o qual sinaliza ao nível Supervisores Modulares a ocorrência do evento. Os procedimentos operacionais dirigem a execução das atividades enviando sinais para os atuadores (saídas do sistema de controle) e recebendo informações dos sensores (entradas do sistema de controle).

Cada supervisor segue a seqüência de eventos tratada no nível Sistema Produto. Com base nesta seqüência de eventos o conjunto de supervisores estabelece os sinais de desabilitação, os quais são enviados para o nível Sistema Produto. Um evento controlável só pode ser gerado se o sinal de desabilitação correspondente não está ativado.

5. Método para implementação concentrada do controle supervisório

Dos fatos apresentados no Capítulo 1, verifica-se que a prática industrial corrente para implementação do controle de sistemas industriais é, em geral, realizada de forma empírica. Nos Capítulos 2 e 3 foram apresentados formalismos matemáticos que permitem realizar a modelagem do comportamento de sistemas a eventos discretos e a síntese de controladores. No Capítulo 4 foi apresentada uma arquitetura para o controle de tais sistemas. Foram discutidos, ainda, os principais problemas que surgem na implementação do controle.

Este capítulo apresenta um método para implementação em CLP da arquitetura de controle proposta por QUEIROZ e CURY (2002b). A adoção deste método permite realizar o desenvolvimento do programa de aplicação do CLP de forma sistemática, com base nos resultados obtidos da aplicação de métodos formais de modelagem do sistema e síntese do controlador.

Na seção inicial deste capítulo é apresentada a relevância do método de implementação. Na segunda seção apresenta-se uma visão geral do método, contemplando a estrutura do programa de aplicação resultante da sua adoção, bem como, os modos de operação estabelecidos pelo referido programa. Nas seções seguintes são detalhados os diversos POUs que constituem o programa de aplicação. Ao final do capítulo é realizada uma discussão geral sobre o método.

5.1 A relevância do método de implementação

Em (QUEIROZ e CURY, 2002b) e (QUEIROZ, 2004) é apresentada uma abordagem simplificada da célula de manufatura apresentada no exemplo motivador introduzido na Seção 3.3. É apresentada, também, uma implementação da arquitetura de controle supervisório proposta por estes autores. Contudo, esta implementação apresenta as seguintes limitações:

i) Os modelos adotados para representar o comportamento independente dos subsistemas constituem um caso particular de modelagem. Não são apresentadas orientações de como implementar os módulos do sistema produto correspondentes a modelos mais genéricos. A mesma

limitação se aplica para os procedimentos operacionais que detalham as atividades a serem executadas por tais subsistemas;

ii) A implementação é realizada na linguagem *Ladder Diagram*. Apesar desta linguagem ser amplamente difundida entre os programadores de CLP ela é de baixo nível, o que dificulta a representação de estruturas genéricas;

iii) O programa de aplicação do CLP é organizado em um único *Program*, o que dificulta a interpretação, modificação e reaproveitamento do código. Este fato também compromete a modularidade do sistema e dificulta a distribuição do controle;

iv) A implementação não especifica procedimentos para :

- conduzir o sistema para o estado inicial através de uma seqüência de tarefas pré-estabelecida;
- reação em casos de emergência;
- interrupção da geração de eventos controláveis;
- geração seletiva de eventos controláveis.

O método de implementação apresentado neste capítulo visa superar tais limitações. Para tanto o método apresentado é baseado em linguagens de alto nível (*Sequential Function Chart* e *Structured Text*); o programa de aplicação resultante é estruturado (e modularizado) em diversos *Function Blocks*; são previstos diversos modos de operação do sistema. Além disto, o método de implementação é aplicável para modelos genéricos de representação do comportamento do sistema a ser controlado.

5.2 Visão geral do método de implementação

A aplicação do método para implementação em CLP da arquitetura de controle proposta por QUEIROZ e CURY (2002b) requer os seguintes elementos:

– Uma representação por sistema produto ($\{\mathbf{G_i} \mid i \in I\}$) que descreve o comportamento livre do sistema a ser controlado, com $\Sigma_c = \cup_{\forall i \in I} \Sigma_c^{Gi}$, $\Sigma_{uc} = \cup_{\forall i \in I} \Sigma_{uc}^{Gi}$ e $\Sigma = \Sigma_c \cup \Sigma_{uc}$;

– Um conjunto de procedimentos operacionais ($\{\mathbf{o_\sigma} \mid \sigma \in \Sigma\}$) que detalha as atividades a serem realizadas pelo sistema a ser controlado;

– Um conjunto de supervisores ($\{\mathfrak{S}_j \mid j \in J\}$) que coordena a operação concorrente dos diversos subsistemas que compõem o sistema a ser controlado.

Com a aplicação do método, cada elemento dos conjuntos $\{\mathfrak{S}_j \mid j \in J\}$, $\{\mathbf{G_i} \mid i \in I\}$ e $\{\mathbf{o_\sigma} \mid \sigma \in \Sigma\}$ é representado através de um SFC, o que resulta nos respectivos conjuntos de SFCs

$\{s_j \mid j \in J\}$, $\{g_i \mid i \in I\}$ e $\{o_\sigma \mid \sigma \in \Sigma\}$. Cada um destes SFCs constitui a seção de código de um FB homônimo. O método de implementação define procedimentos sistemáticos que permitem converter cada autômato em $\{\mathbf{G_i} \mid i \in I\}$ e cada supervisor em $\{\mathcal{S}_j \mid j \in J\}$ no SFC correspondente. Tais procedimentos especificam as variáveis empregadas no programa de aplicação. Considera-se que a obtenção de um SFC que representa um procedimento operacional é parte do processo de modelagem do sistema, não havendo um procedimento sistemático para tanto, apenas orientações gerais sobre o emprego de algumas variáveis especificadas nos procedimentos mencionados.

Sempre que ocorre o tratamento de um evento de um subsistema em $\{\mathbf{G_i}, i \in I\}$, deve ser suspenso momentaneamente o tratamento de eventos de todo subsistema que compartilha alguma célula de controle com o subsistema em questão. Para implementar esta ação, a cada subsistema em $\{\mathbf{G_i}, i \in I\}$ é associado um FB denominado dg_i. Cada FB em $\{dg_i, i \in I\}$ é o responsável por autorizar ou suspender o tratamento de eventos do subsistema correspondente. Na Seção 5.4 são apresentados maiores detalhes sobre a suspensão no tratamento de eventos.

Há três estruturas lógicas auxiliares que permitem realizar a chamada seqüencial dos FBs em $\{s_j \mid j \in J\}$, $\{g_i \mid i \in I\}$, $\{dg_i \mid i \in I\}$, e $\{o_\sigma \mid \sigma \in \Sigma\}$. Tais estruturas são os FBs MS, PS e OP. O FB MS juntamente com os FBs em $\{s_j \mid j \in J\}$ constituem o nível Supervisores Modulares da Arquitetura de Controle Supervisório de Queiroz e Cury. O nível Sistema Produto da referida arquitetura de controle é constituído pelo FB PS juntamente com os conjuntos de FBs em $\{dg_i \mid i \in I\}$ e $\{g_i \mid i \in I\}$. O FB OP junto com os FBs em $\{o_\sigma \mid \sigma \in \Sigma\}$ constituem o nível Procedimentos Operacionais.

O programa de aplicação resultante da adoção deste método é constituído de um *Program* e diversos *Function Blocks*. A Tabela 5.1 detalha a estrutura do programa de aplicação.

A Figura 5.1 apresenta o SFC Main, o qual constitui a seção de código do *Program* Main. O passo ativo deste SFC estabelece o modo de operação do sistema. Há seis modos de operação distintos: *Software Initialization* (passo SI); *Physical System Initialization* (passo PSI); *Manual* (passo Man); *Supervised* (passo Sup); *Emergency* (passo Emg) e *Idle* (passos init e PSIted). Estes modos de operação permitem a coordenação manual ou supervisionada do sistema a ser controlado, bem como a execução de procedimentos que conduzem o sistema a ser controlado para o estado inicial e a execução de procedimentos de emergência.

Quando o sistema está no modo *Supervised* o conjunto de supervisores coordena a operação concorrente dos diversos subsistemas. A ação de controle do conjunto de supervisores estabelece o estado de cada sinal de desabilitação (ver Figura 4.13).

O operador é o responsável pela coordenação dos diversos subsistemas quando o sistema está no modo *Manual*. Por padrão, neste modo de operação, todos os sinais de desabilitação são

ativados. Isto inibe a geração de qualquer evento controlável. O operador pode conduzir o sistema realizando a desativação seletiva dos sinais de desabilitação. Este modo de operação é particularmente útil para a verificação da modelagem e implementação dos procedimentos operacionais. Ele também permite que o operador conduza o sistema para um estado de interesse através de uma determinada seqüência de eventos. Destaca-se que, a ação do operador pode conduzir o sistema para uma situação que viola alguma especificação de controle, isto é, a seqüência de eventos ($s \in \Sigma^*$) desenvolvida pelo sistema não pertence ao comportamento ótimo sob supervisão ($s \notin \sup\mathfrak{C}(K)$, onde K é a linguagem que marca o comportamento desejado para o sistema a ser controlado). Enquanto a seqüência de eventos permanece dentro deste comportamento (a variável Booleana "safe" armazena valor lógico VERDADEIRO), é possível alterar diretamente para o modo *Supervised*. Caso este comportamento tenha sido violado, o operador é notificado do fato e o modo *Supervised* não pode ser ativado sem antes passar pelo modo *Software Initialization*.

Tabela 5.1 – *Program Organization Units* do programa de aplicação

nome	tipo	comentários
s_j $j \in J$	FB	Representa o supervisor $\mathfrak{S}_j$
MS	FB	Estrutura lógica que realiza a chamada ordenada de todos os FBs em $\{s_j \mid j \in J\}$
g_i $i \in I$	FB	Representa o módulo do sistema produto associado ao subsistema $\mathbf{G_i}$
dg_i $i \in I$	FB	Estrutura lógica que suspende ou autoriza o tratamento de eventos em Σ^{Gi}
PS	FB	Estrutura lógica que realiza a chamada ordenada de todos os FBs em $\{g_i \mid i \in I\}$ e $\{dg_i \mid i \in I\}$
o_σ $\sigma \in \Sigma$	FB	Representa o procedimento operacional $\mathbf{o_\sigma}$
OP	FB	Estrutura lógica que realiza a chamada ordenada de todos os FBs em $\{o_\sigma \mid \sigma \in \Sigma\}$
Main	*Program*	Coordena a operação da arquitetura de controle

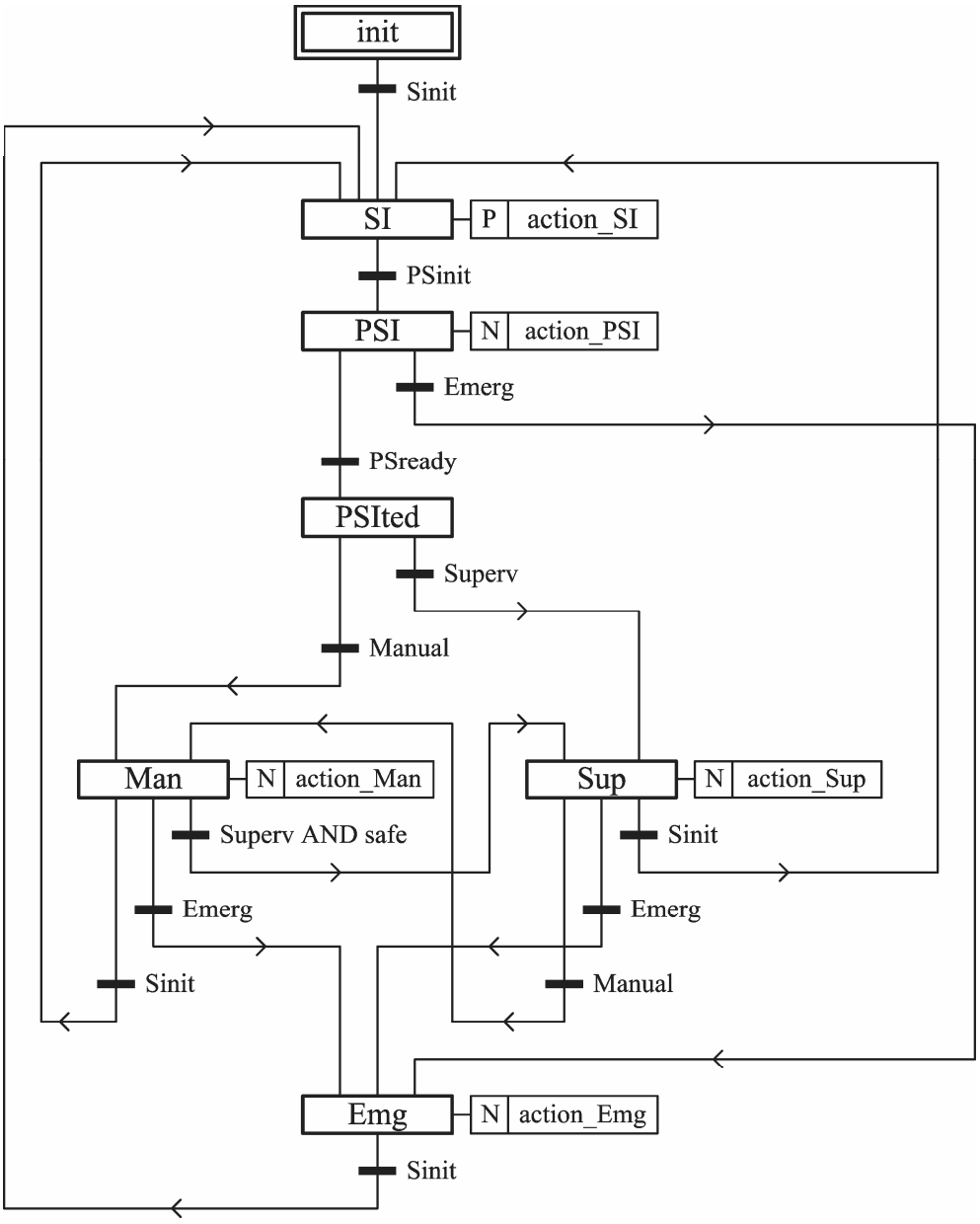


Figura 5.1 – SFC Main

Quando o passo SI do SFC Main é ativado, o sistema entra no modo *Software Initialization* e a ação associada a este passo (action_SI) é executada apenas uma vez. Isto inicializa todos os SFCs nos conjuntos $\{s_j \mid j \in J\}$, $\{g_i \mid i \in I\}$ e $\{o_\sigma \mid \sigma \in \Sigma\}$. Isto também estabelece a todas as variáveis do CLP o valor inicial apropriado.

Enquanto o passo PSI do SFC Main está ativo, o sistema está no modo *Physical System Initialization* e a ação action_PSI é executada uma vez a cada ciclo de atualização do CLP. Esta ação deve ser projetada de forma a conduzir o sistema a ser controlado para o estado inicial. Quando o sistema a ser controlado alcança este estado (a variável Booleana "PSready" assume

valor lógico VERDADEIRO), o passo PSIted do SFC Main é ativado. Enquanto este passo está ativo o sistema está no modo *Idle*.

Finalmente, no modo *Emergency* (passo Emg ativo) todas as atividades são imediatamente suspensas e procedimentos de emergência podem ser executados pela ação action_Emg.

No SFC Main o estado das variáveis Booleanas empregadas nas condições de transição (Sinit, PSinit, Emerg, Manual e Superv) podem ser determinados pela transição positiva de sinal de variáveis de entrada do CLP ou através de uma interface com o usuário desenvolvida em um sistema de supervisão.

5.3 SFCs e FBs no conjunto $\{g_i \mid i \in I\}$

Esta seção apresenta o procedimento sistemático que permite converter cada autômato em $\{\mathbf{G_i} \mid i \in I\}$ no SFC correspondente. Tais SFCs constituem a seção de código dos FBs em $\{g_i \mid i \in I\}$. A *etapa-1* deste procedimento define as instâncias de tais FBs bem como as variáveis envolvidas nas respectivas seções de código. A *etapa-2* permite converter o autômato $\mathbf{G_i}$, $i \in I$, em outro autômato que satisfaz um conjunto de propriedades de interesse. A *etapa-3* define o SFC g_i com base no autômato obtido na *etapa-2*.

etapa-1) Definição de variáveis do CLP e das instâncias dos FBs

Duas variáveis Booleanas, denominadas "$g_i$evt" e "g_id", são associadas a cada subsistema em $\{\mathbf{G_i}, i \in I\}$. A ativação (no FB g_i) da variável $g_i$evt sinaliza que foi tratado algum evento em Σ^{Gi}. O FB dg_i autoriza ou suspende o tratamento de eventos em Σ^{Gi} através da desativação ou ativação da variável g_id correspondente. A decisão de autorizar ou suspender o tratamento de eventos é estabelecida com base no estado lógico das variáveis em $\{g_i\text{evt} \mid i \in I\}$.

Uma variável Booleana, denominada "σ" $\in E_i$, é associada a cada evento $\sigma \in \Sigma^{Gi}$, $i \in I$. Duas outras variáveis Booleanas são associadas a cada evento controlável $\sigma \in \Sigma_c{}^{Gi}$, $i \in I$. Tais variáveis são denominadas "σd" $\in D_i$ e "cmdσ" $\in C_i$. Uma variável do tipo *Unsigned Integer*, denominada "rspσ" $\in R_i$, é associada a cada evento não-controlável $\sigma \in \Sigma_{uc}{}^{Gi}$, $i \in I$. Os quatro conjuntos de variáveis assim obtidos são tais que $|E_i| = |\Sigma^{Gi}|$, $|D_i| = |C_i| = |\Sigma_c{}^{Gi}|$ e $|R_i| = |\Sigma_{uc}{}^{Gi}|$.

A ativação das variáveis σ, σd e cmdσ é realizada, respectivamente, para sinalizar o tratamento do evento $\sigma \in \Sigma^{Gi}$; para ativar o sinal de desabilitação e o comando associados ao evento controlável $\sigma \in \Sigma_c{}^{Gi}$. O valor armazenado na variável rspσ indica o número de ocorrências do evento não-controlável $\sigma \in \Sigma_{uc}{}^{Gi}$ que estão pendentes para tratamento.

Além destas, deve ser criada uma variável Booleana denominada "CED". Como será observado na *etapa-3* deste procedimento, cada transição do SFC g_i, $i \in I$, é associada a um evento em Σ^{Gi}. A ativação da variável CED suspende o tratamento de todo evento controlável, pois torna falsa toda condição de transição associada a evento controlável. A adequada manipulação desta variável (ativação e desativação nas ações associadas aos passos Sup (action_Sup) e Man (action_Man) do SFC Main) permite priorizar o tratamento de eventos não-controláveis em detrimento do tratamento de eventos controláveis.

Cada FB g_i, $i \in I$, deve possuir exatamente uma instância, denominada "Inst_g_i".

A Tabela 5.2 apresenta o escopo de definição de tais variáveis e FBs nos diferentes POUs do programa de aplicação. Nenhuma destas variáveis precisa ser associada com variáveis de entrada ou saída do CLP.

♦ fim da etapa 1 ♦

etapa-2) Conversão do autômato $\mathbf{G_i}$ no autômato $\mathbf{H_i}$, $i \in I$

O objetivo principal do SFC g_i, $i \in I$, é realizar o tratamento de eventos em Σ^{Gi}. Em um SFC, é possível associar as ações a cada um de seus passos, nunca a suas transições. Assim, cada passo do SFC g_i deve estar associado a um evento distinto em Σ^{Gi}. Visto que o procedimento para obtenção destes SFCs associa cada passo a um estado do autômato correspondente, o objetivo desta etapa é obter um autômato $\mathbf{H_i}$ correspondente ao autômato $\mathbf{G_i}$, tal que, cada estado do autômato $\mathbf{H_i}$ esteja associado a um evento distinto em Σ^{Gi}. Além disto, em $\mathbf{H_i}$ não devem haver auto-laços. Isto é necessário para garantir que, para toda transição do SFC g_i o passo sucessor seja sempre diferente do passo antecessor da transição em questão.

Caso exista algum auto-laço no autômato $\mathbf{G_i} = (\Sigma^{Gi}, Q^{Gi}, \delta^{Gi}, q0^{Gi}, Q_m{}^{Gi})$, isto é, ($\exists\ q \in Q^{Gi}$ e $\sigma \in \Sigma^{Gi}$) tal que $\delta^{Gi}(q, \sigma) = q$, então, antes de obter o autômato $\mathbf{H_i}$ deve ser obtido um autômato equivalente ao autômato $\mathbf{G_i}$ no qual não há qualquer auto-laço. Este autômato sem qualquer auto-laço passa a ser considerado como o autômato $\mathbf{G_i}$ no restante deste procedimento. Se o autômato $\mathbf{G_i}$ não possui auto-laços, então o autômato $\mathbf{H_i}$ não possuirá auto-laços.

Tabela 5.2 – Tipos de variáveis nos diferentes POUs		
Instância de variáveis e FBs	Escopo de definição	POU
CED	*Global*	Program Main
	External	todo FB em $\{g_i \mid i \in I\}$
$g_i$evt ($i \in I$)	*Global*	Program Main
	External	FB g_i correspondente
		FB dg_i correspondente
		todo dg_z, $z \in I$, tal que $\mathbf{G_i}$ e $\mathbf{G_z}$ compartilham alguma célula de controle em comum
g_id ($i \in I$)	*Global*	Program Main
	External	FB dg_i correspondente
		FB PS
$\sigma \in E_i$ ($i \in I$)	*Global*	Program Main
	External	FB g_i correspondente
		todo FB s_j, $j \in J$, tal que $\mathbf{G_i}$ e $\mathcal{J}_j$ compartilham alguma célula de controle em comum
$\sigma d \in D_i$ ($i \in I$)	*Global*	Program Main
	External	FB g_i correspondente
		FB MS
cmd$\sigma \in C_i$ ($i \in I$)	*Global*	Program Main
	External	FB g_i correspondente
		FB o_σ sempre que cmdσ é empregado em alguma condição de transição do SFC o_σ
rsp$\sigma \in D_i$ ($i \in I$)	*Global*	Program Main
	External	FB g_i correspondente
		FB o_σ sempre que rspσ é ativada em algum passo do SFC o_σ
Inst_g_i ($i \in I$)	*Global*	Program Main
	External	FB PS

Exemplo 5.1)

Os autômatos **G1** e **G2** apresentados na Figura 5.2 são equivalentes, isto é, $L(\mathbf{G1}) = L(\mathbf{G2})$ e $L_m(\mathbf{G1}) = L_m(\mathbf{G2})$. O autômato **G1** possui um auto-laço no estado 1, pois $\delta^{G1}(1, \lambda) = 1$, já o autômato **G2** não possui qualquer auto-laço.

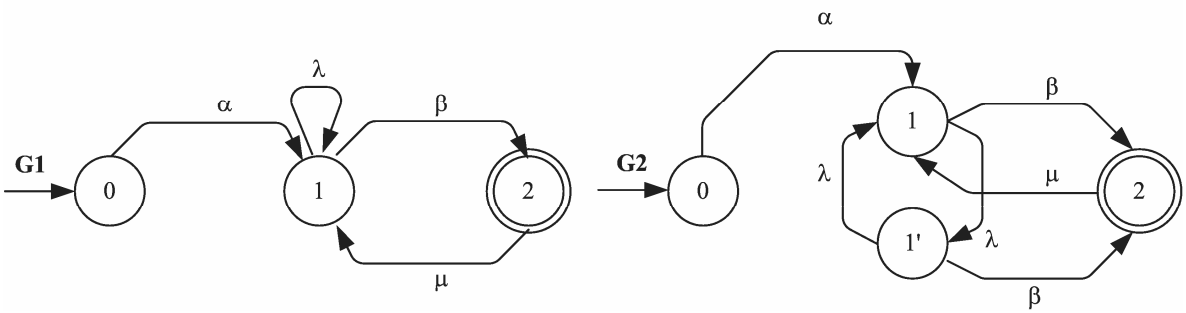


Figura 5.2 – Autômatos equivalentes, com auto-laço e sem auto-laço

♦ fim do exemplo 5.1 ♦

Considere cada autômato $\mathbf{G_i} = (\Sigma^{Gi}, Q^{Gi}, \delta^{Gi}, q0^{Gi}, Q_m{}^{Gi})$ na representação por sistema produto $\{\mathbf{G_i} \mid i \in I\}$ e cada autômato correspondente $\mathbf{H_i} = (\Sigma^{Hi}, Q^{Hi}, \delta^{Hi}, q0^{Hi}, Q_m{}^{Hi})$, onde:

Σ^{Hi} – alfabeto de eventos, $\Sigma^{Hi} = \Sigma^{Gi}$;

$q0^{Hi}$ – estado inicial;

Q^{Hi} – conjunto de estados, com $Q^{Hi} \subseteq \{Q^{Gi} \times \Sigma^{Gi}\} \cup \{q0^{Hi}\}$;

$\delta^{H}{}_{i}$ – função de transição de estados, $\delta^{Hi} : Q^{Hi} \times \Sigma^{Hi} \rightarrow Q^{Hi}$;

$Q_m{}^{Hi}$ – conjunto de estados marcados.

A conversão do autômato $\mathbf{G_i}$, $i \in I$, no autômato $\mathbf{H_i}$ é realizada para que este último satisfaça as seguintes propriedades:

i) $L(\mathbf{H_i}) = L(\mathbf{G_i})$;

ii) $[\ \forall(q1, q2 \in Q^{Hi}), \forall(\sigma a, \sigma b \in \Sigma^{Hi})\ ]$ $\quad \delta^{Hi}(q1, \sigma a) = \delta^{Hi}(q2, \sigma b)$ somente se $\sigma a = \sigma b$;

iii) $[\ \forall(q \in Q^{Hi}), \forall(\sigma \in \Sigma^{Hi})\ ]$ $\quad \delta^{Hi}(q, \sigma) \neq q0^{Hi}$;

iv) $[\ \forall(q \in Q^{Hi}), \forall(\sigma \in \Sigma^{Hi})\ ]$ $\quad \delta^{Hi}(q, \sigma) \neq q$.

A propriedade (*i*) estabelece que a seqüência de eventos gerada pelo autômato $\mathbf{H_i}$ é exatamente igual à seqüência gerada pelo autômato $\mathbf{G_i}$, isto é, com exceção da capacidade de reconhecer tarefas completas, os dois autômatos descrevem da mesma forma o comportamento independente do subsistema associado a estes autômatos. A propriedade (*ii*) estabelece que toda

transição que conduz a um determinado estado do autômato $\mathbf{H_i}$ é sempre devida à ocorrência do mesmo evento. A propriedade (*iii*) estabelece que a ocorrência de um evento qualquer nunca conduz ao estado inicial do autômato $\mathbf{H_i}$. A propriedade (*iv*) estabelece que o autômato $\mathbf{H_i}$ não possui auto-laços. Esta propriedade só é satisfeita se o autômato $\mathbf{G_i}$ não possui auto-laços.

O procedimento de conversão apresentado a seguir é baseado no procedimento proposto em (CARROLL e LONG, 1989) para converter um autômato de Mealy em um autômato de Moore. O autômato $\mathbf{H_i}$ é obtido tomando a componente acessível do autômato definido como:

i) $q0 = (q0^{Gi}, \sigma_{dummy})$, tal que $\sigma_{dummy} \notin \Sigma$;

ii) $[\ \forall(q \in Q^{Gi}), \forall(\sigma1 \in \Sigma^{Gi}), \forall(\sigma2 \in \Sigma^{Gi} \cup \{\sigma_{dummy}\})\ ]$ $\qquad \delta((q, \sigma2), \sigma1) = (\delta^{Gi}(q, \sigma1), \sigma1)$;

iii) $Q_m = \varnothing$.

Visto que o estado $q \in Q^{Hi}$, tal que $q \neq q0^{Hi}$ com $i \in I$, é um par ordenado em $\{Q^{Gi} \times \Sigma^{Gi}\}$, pode-se afirmar que tal estado é associado com o evento $\sigma \in \Sigma^{Gi}$ que constitui o segundo elemento de tal par ordenado.

A definição do item (*iii*) é baseada no fato de que, apesar da marcação de estados dos autômatos empregados na descrição do comportamento independente dos subsistemas que compõem o sistema a ser controlado e das especificações de controle ser crucial na síntese dos supervisores, não consideramos tal informação na obtenção dos SFCs em $\{g_i \mid i \in I\}$. A marcação de estados só deveria ser considerada se o sistema de controle devesse ser capaz de reconhecer e sinalizar o tratamento de eventos que resultam na conclusão de tarefas.

♦ fim da etapa 2 ♦

etapa-3) Definição do SFC $g_i = (X^{gi}, T^{gi}, x0^{gi})$, $i \in I$

Esta etapa é baseada no procedimento apresentado em (CASSANDRAS e LAFORTUNE, 1999) para conversão de autômatos em Redes de Petri.

O SFC g_i, $i \in I$, é obtido realizando os itens (*i*) a (*iii*) abaixo:

i) Um passo $xq \in X^{gi}$ é associado a cada estado $q \in Q^{Hi}$, de forma que $|X^{gi}| = |Q^{Hi}|$. Com isto cada passo do SFC fica automaticamente associado ao evento que está associado ao estado correspondente;

ii) $x0^{gi}$ é o passo associado ao estado $q0^{Hi}$;

iii) Uma transição $(xq, xq') \in T^{gi}$ é associada a cada tripla-ordenada (q, σ, q') em $\mathbf{H_i}$, onde $\delta^{Hi}(q, \sigma) = q'$ e os passos $xq, xq' \in X^{gi}$ são respectivamente associados aos estados $q, q' \in Q^{Hi}$. Esta associação deve ser realizada de forma que $|T^{gi}|$ iguala o número de triplas-ordenadas acima

mencionadas. Este item determina que cada transição em T^{gi} fica automaticamente associada ao evento relacionado na tripla-ordenada correspondente.

Este procedimento não estabelece prioridade à avaliação de transições no caso de seleção de seqüências divergentes.

A condição de transição associada a cada transição em T^{gi}, $i \in I$, é a expressão Booleana "NOT σd AND NOT CED" ou a expressão Booleana "rspσ > 0". Estas expressões assumem que $\sigma \in \Sigma^{Gi}$ é o evento associado à transição em questão. Além disto, $\sigma \in \Sigma^{Gi}$ é o evento associado às variáveis $\sigma d \in D_i$ ou $rsp\sigma \in R_i$. A primeira expressão deve ser empregada sempre que o referido evento é controlável ($\sigma \in \Sigma_c^{Gi}$) e a segunda expressão sempre que o evento é não-controlável ($\sigma \in \Sigma_{uc}^{Gi}$).

A ação associada a cada passo em $\{ X^{gi} - \{x0^{gi}\} \}$, $i \in I$, é a apresentada na Figura 5.3a ou 5.3b. Nenhuma ação é associada ao passo inicial $x0^{gi}$. Estas ações assumem que $\sigma \in \Sigma^{Gi}$ é o evento associado ao passo em questão. Além disto, $\sigma \in \Sigma^{Gi}$ é o evento associado às variáveis $\sigma \in E_i$, $cmd\sigma \in C_i$ e $rsp\sigma \in R_i$. A ação apresentada na Figura 5.3a deve ser empregada sempre que o referido evento é controlável ($\sigma \in \Sigma_c^{Gi}$) e a apresentada na Figura 5.3b sempre que o evento é não-controlável ($\sigma \in \Sigma_{uc}^{Gi}$). Estas ações assumem ainda que o subsistema $\mathbf{G_i}$ é associado à variável $g_i$evt.

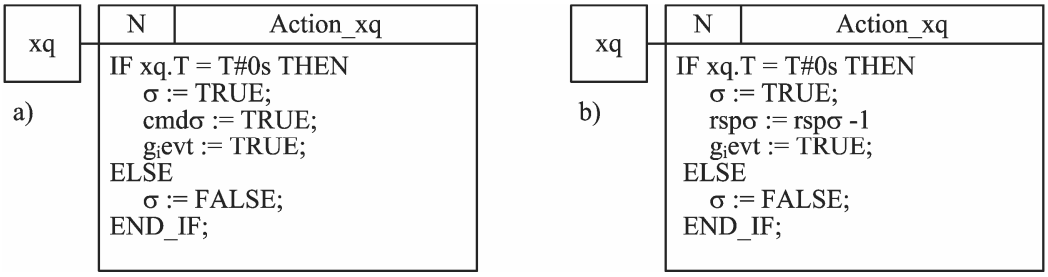


Figura 5.3 – Ações associadas ao passo xq do SFC g_i, $i \in I$

♦ fim da etapa-3 ♦

Exemplo 5.2)

Considere o sistema de furação do exemplo motivador introduzido na Seção 3.3, cujo comportamento independente é representado pelo autômato $\mathbf{G_2}$ (Figura 3.4). Para este subsistema tem-se que $\Sigma_c^{G2} = \{\alpha 2\}$ e $\Sigma_{uc}^{G2} = \{\beta 2, \lambda 2, \mu 2\}$.

De acordo com a *etapa-1*, as variáveis g2evt e g2d devem ser associadas ao subsistema $\mathbf{G_2}$. Representa-se esta associação como ($\mathbf{G_2}$, g2evt) e ($\mathbf{G_2}$, g2d). É necessário realizar também a associação "evento *x* variável do CLP" apresentada a seguir: (α2, a2), (α2, cmda2), (α2, a2d),

(β2, b2), (β2, rspb2), (λ2, l2), (λ2, rspl2), (μ2, m2), (μ2, rspm2), tal que E_2 = {a2, b2, l2, m2}, D_2 = {a2d}, C_2 = {cmda2} e R_2 = {rspb2, rspl2, rspm2}.

A realização da *etapa-2* resulta no autômato **H_2** apresentado na Figura 5.4.

Sendo Q^{G2} = {0, 1, 2} e Σ^{G2} = {α2, β2, λ2, μ2}, tem-se que $Q^{H2} \subseteq Q^{G2}$ x Σ^{G2} = { (0, α2), (0, β2), (0, λ2), (0, μ2), (1, α2), (1, β2), (1, λ2), (1, μ2), (2, α2), (2, β2), (2, λ2), (2, μ2) } ∪ {$q0^{H2}$}, com $q0^{H2} = (0, \sigma_{dummy})$. A função de transição de estados $\delta^{H2} : Q^{H2} \times \Sigma^{H2} \rightarrow Q^{H2}$ é obtida por:

$\delta^{H2}((0, \sigma_{dummy}), \alpha2) = (\delta^{G2}(0, \alpha2), \alpha2) = (1, \alpha2)$;

$\delta^{H2}((1, \alpha2), \beta2) = (\delta^{G2}(1, \beta2), \beta2) = (0, \beta2)$;

$\delta^{H2}((1, \alpha2), \lambda2) = (\delta^{G2}(1, \lambda2), \lambda2) = (2, \lambda2)$;

$\delta^{H2}((0, \beta2), \alpha2) = (\delta^{G2}(0, \alpha2), \alpha2) = (1, \alpha2)$;

$\delta^{H2}((2, \lambda2), \mu2) = (\delta^{G2}(2, \mu2), \mu2) = (0, \mu2)$;

$\delta^{H2}((0, \mu2), \alpha2) = (\delta^{G2}(0, \alpha2), \alpha2) = (1, \alpha2)$;

Na determinação da função $\delta((q, \sigma2), \sigma1) = (\delta^{G2}(q, \sigma1), \sigma1)$, sempre que $\delta^{G2}(q, \sigma1)$ não está definida então $\delta((q, \sigma2), \sigma1)$ não está definida. Por exemplo, $\delta((0, \sigma_{dummy}), \beta2) = (\delta^{G2}(0, \beta2), \beta2)$. Como $\delta^{G2}(0, \beta2)$ não está definida, então $\delta((0, \sigma_{dummy}), \beta2)$ também não está definida.

Da aplicação do procedimento descrito anteriormente para obter a função de transição de estados verifica-se que esta última está definida a partir de estados não-acessíveis. Por exemplo $\delta((0, \lambda2), \alpha2) = (\delta^{G2}(0, \alpha2), \alpha2) = (1, \alpha2)$. Porém, $(0, \lambda2) \in Q^{G2} \times \Sigma^{G2}$ é um estado não-acessível, sendo excluído quando se toma componente acessível do autômato correspondente, assim $(0, \lambda2) \notin Q^{H2}$.

Por racionalidade de espaço omite-se a determinação da função de transição de estados δ^{H2} para todos os demais pares ordenados em $Q^{H2} \times \Sigma^{H2}$.

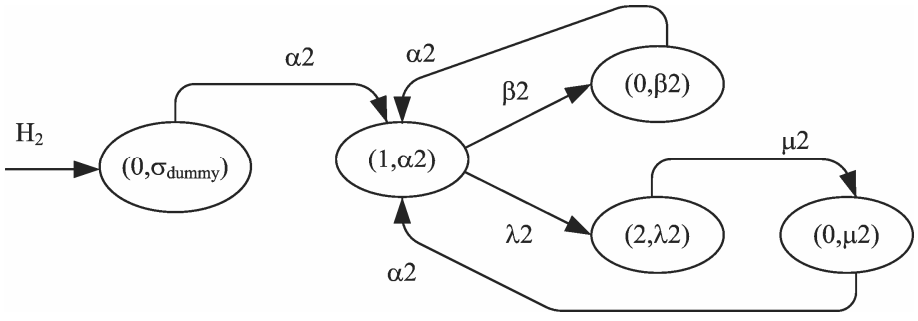


Figura 5.4 – Autômato **H_2** correspondente ao autômato **G_2**

Como pode ser observado na Figura 5.4, há três transições que conduzem ao estado (1, $\alpha 2$) deste autômato. Conforme estabelecido pela propriedade (*ii*) apresentada na *etapa-2*, todas elas são devidas à ocorrência do mesmo evento, no caso, o evento $\alpha 2$. Conforme a propriedade (*iii*) não há qualquer transição conduzindo ao estado inicial, também não há qualquer auto-laço (propriedade (*iv*)). Têm-se, ainda que, $L(\mathbf{G_2}) = L(\mathbf{H_2})$ (propriedade (*i*)).

A realização da *etapa-3* resulta no SFC g_2 apresentado na Figura 5.5. Este SFC foi obtido realizando a associação "estado de $\mathbf{H_2}$ *x* passo de g_2" apresentada a seguir: $((0, \sigma_{dummy}), x0)$, $((1, \alpha 2), x1)$, $((0, \beta 2), x2)$, $((2, \lambda 2), x3)$, $((0, \mu 2), x4)$. Desta associação vê-se que o passo x1 está associado ao evento $\alpha 2$, que o passo x2 está associado ao evento $\beta 2$, que os passos x3 e x4 estão associados, respectivamente, aos eventos $\lambda 2$ e $\mu 2$. O passo x0 está associado ao evento σ_{dummy}, o qual não é empregado na representação por sistema produto ($\sigma_{dummy} \notin \Sigma$).

Para obter o SFC g2 também foi realizada a associação "tripla ordenada (q, σ, q') *x* transição (xq, xq')" apresentada a seguir:

- ($(0, \sigma_{dummy})$, $\alpha 2$, $(1, \alpha 2)$) com (x0, x1);

- ($(1, \alpha 2)$, $\beta 2$, $(0, \beta 2)$) com (x1,x2);

- ($(1, \alpha 2)$, $\lambda 2$, $(2, \lambda 2)$) com (x1, x3);

- ($(0, \beta 2)$, $\alpha 2$, $(1, \alpha 2)$)com (x2, x1);

- ($(2, \lambda 2)$, $\mu 2$, $(0, \mu 2)$) com (x3, x4);

- ($(0, \mu 2)$, $\alpha 2$, $(1, \alpha 2)$) com (x4, x1).

A seção de declaração de variáveis do FB g2 deve ser como apresentado na Figura 5.6. Não são incluídas as variáveis associadas aos passos do SFC (as variáveis na forma step_name.X e step_name.T) por serem automaticamente criadas pelo sistema de programação do CLP. O SFC apresentado na Figura 5.5 constitui a seção de código deste FB.

Considere o passo x1 do SFC g_2, o qual está associado ao evento controlável $\alpha 2 \in \Sigma_c^{G2}$. Conforme estabelecido pelas transições (x0, x1), (x2, x1) e (x4, x1), para que ocorra a ativação do passo x1 é necessário que a ocorrência deste evento não esteja desabilitada por qualquer supervisor (NOT a2d) e que o tratamento de eventos controláveis não esteja suspenso (NOT CED). Também é necessário que a ocorrência deste evento seja prevista no modelo que descreve o comportamento independente deste subsistema (qualquer dos passos x0 ou x2 ou x4 esteja ativo). Na ativação deste passo (x1.T = T#0s resulta VERDADEIRO) ocorre a geração do evento $\alpha 2$ e a ativação do comando associado a este evento (a2 := TRUE e cmda2 := TRUE); é sinalizado que houve o tratamento de algum evento em Σ^{G2} (g2evt := TRUE).

Considere agora o passo x2, o qual está associado ao evento não-controlável $\beta 2 \in \Sigma_{uc}^{G2}$. Para a ativação deste passo é necessário que exista alguma ocorrência do evento β2 pendente para ser tratada (rspb2 > 0). É necessário, também, que a ocorrência deste evento seja prevista no modelo que descreve o comportamento independente deste subsistema (passo x1 ativo). Na ativação deste passo é sinalizada a ocorrência do evento β2 (b2 := TRUE); é sinalizado que houve o tratamento de algum evento em Σ^{G2} (g2evt := TRUE); é computado que foi realizado o tratamento de uma ocorrência do evento β2 (rspb2 := rspb2 - 1).

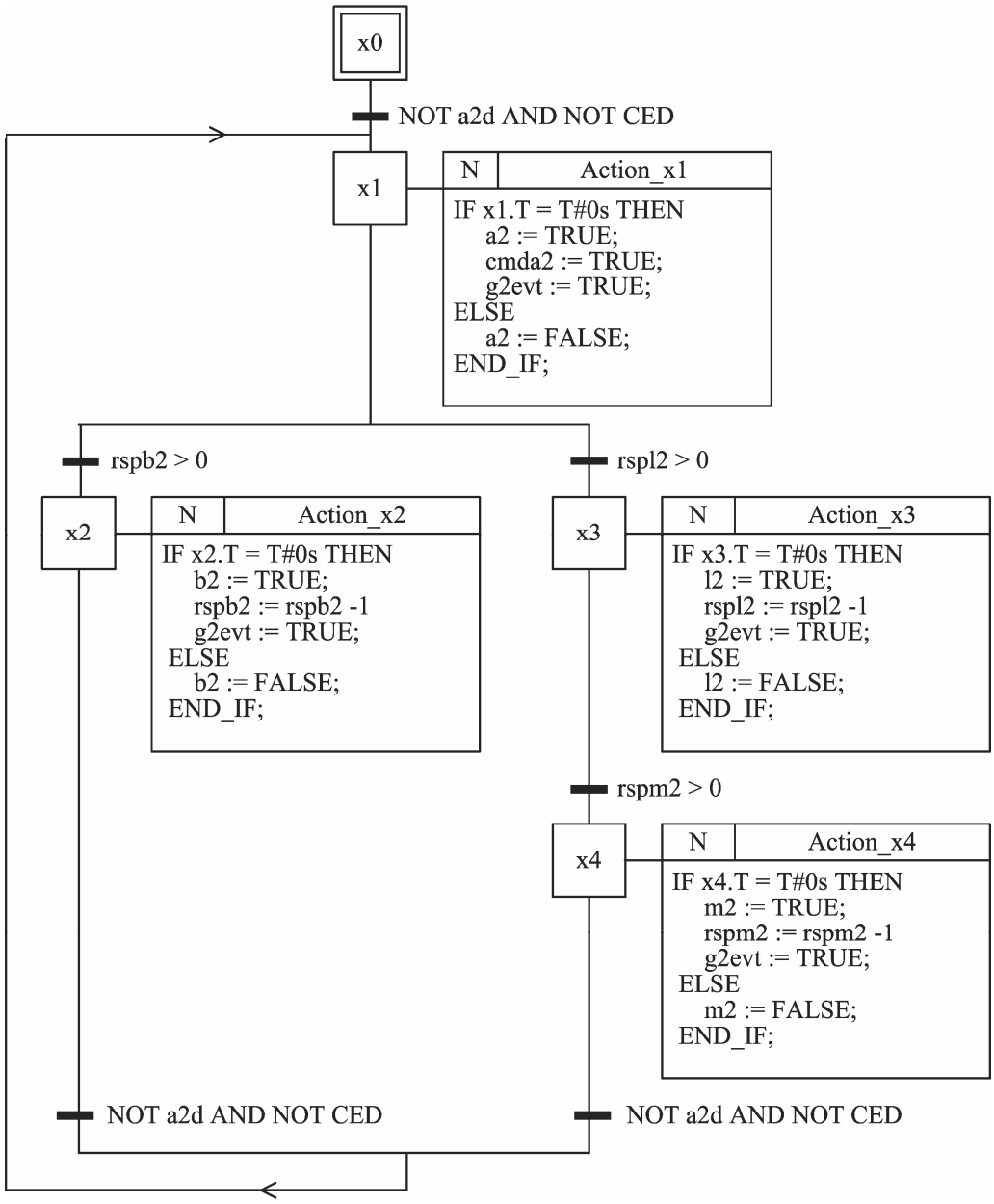


Figura 5.5 – SFC g_2

```
FUNCTION_BLOCK g2
(* Seção de declaração de variáveis *)

VAR_EXTERNAL
   a2, b2, l2, m2: BOOL;                          (* variáveis em E2 *)
   cmda2 : BOOL;                                  (* variável em C2 *)
   a2d : BOOL;                                    (* variável em D2 *)
   rspb2, rspl2, rspm2 : UNSIGNED INTEGER;        (* variáveis em R2 *)
   g2evt : BOOL;                                  (* variável em gievt *)
   CED : BOOL;
END_VAR

(* Fim da seção de declaração de variáveis *)
```

Figura 5.6 – Seção de declaração de variáveis do FB g2

O FB g0 (Figura 4.2), discutido no Exemplo 4.1, foi obtido com a aplicação do procedimento definido nesta seção ao subsistema $\mathbf{G_0}$ do exemplo motivador introduzido na Seção 3.3.

♦ fim do exemplo 5.2 ♦

5.4 Funções lógicas e FBs no conjunto $\{dg_i \mid i \in I\}$

Considere individualmente cada subsistema em $\{\mathbf{G_i} \mid i \in I\}$. O conjunto formado por todos os subsistemas que compartilham alguma célula de controle com o subsistema $\mathbf{G_i}$ considerado pode ser representado por $\{\mathbf{G_k} \mid k \in K_i, K_i \subseteq I\}$. Nesta representação K_i é o conjunto de índices que identificam os subsistemas que compartilham alguma célula de controle com o subsistema $\mathbf{G_i}$ considerado. O exemplo 5.3 ilustra estes conjuntos quando se considera o subsistema G_2.

Conforme discutido no início da Seção 5.2, o tratamento de eventos em Σ^{Gi} deve ser suspenso sempre que algum evento em qualquer Σ^{Gk}, $k \in K_i$, tiver sido tratado. O tratamento de evento em Σ^{Gw}, tal que $w \notin K_i$, não requer a suspensão do tratamento de eventos em Σ^{Gi}. Ou seja, o tratamento de evento de algum subsistema que compartilha alguma célula de controle com $\mathbf{G_i}$ requer que o tratamento de eventos em Σ^{Gi} seja suspenso, porém, o tratamento de eventos de subsistemas que não compartilham qualquer célula de controle com $\mathbf{G_i}$ não requer esta suspensão. A suspensão no tratamento de eventos deve ser mantida até que ocorra a atualização do estado ativo de todo supervisor que compartilha alguma célula de controle com o subsistema que teve algum evento tratado. Esta ação é necessária para evitar o tratamento simultâneo de múltiplos eventos. Além disto, caso o evento tratado seja controlável, a suspensão no tratamento de eventos deve ser mantida até que ocorra a confirmação de que o comando associado a tal evento tenha sido

devidamente processado pelo procedimento operacional correspondente (ver na Seção 5.6 como é realizada tal confirmação). Esta ação é necessária para garantir que ocorra o disparo das atividades associadas ao evento em questão antes que seja realizado o tratamento de outros eventos.

Destaca-se que suspender o tratamento de um evento não-controlável não evita a ocorrência do fenômeno físico associado a tal evento. Isto simplesmente posterga o tratamento da resposta associada a tal evento e a sinalização da ocorrência deste. Por outro lado, suspender o tratamento de um evento controlável, de fato, evita a geração de tal evento e o disparo da execução das atividades associadas ao mesmo.

A seção de código do FB dg_i, $i \in I$, deve estabelecer o valor lógico VERDADEIRO à variável $g_i d$ se pelo menos alguma variável em $\{g_k evt \mid k \in K_i\}$ apresenta valor lógico VERDADEIRO, em caso contrário o valor lógico da variável $g_i d$ deve ser estabelecido como FALSO.

Cada FB dg_i, $i \in I$, deve possuir exatamente uma instância denominada "Inst_dg_i. Tal instância deve ser declarada como *Global* no *Program* Main e *External* no FB PS.

Exemplo 5.3)

Considere o exemplo motivador apresentado na Seção 3.3. Como pode ser observado na Tabela 3.3, o conjunto de subsistemas que compartilham alguma célula de controle com o subsistema $\mathbf{G_2}$ é $\{\mathbf{G_0}, \mathbf{G_1}, \mathbf{G_2}, \mathbf{G_3}\}$, portanto $K_2 = \{0, 1, 2, 3\}$. Com base neste fato, o FB dg2 fica conforme apresentado na Figura 5.7 Este FB assume a associação "subsistema *x* variável do CLP" apresentada a seguir: ($\mathbf{G_0}$, g0evt), ($\mathbf{G_1}$, g1evt), ($\mathbf{G_2}$, g2evt), ($\mathbf{G_3}$, g3evt), ($\mathbf{G_2}$, g2d).

```
FUNCTION_BLOCK dg2

(* Seção de declaração de variáveis *)

VAR_EXTERNAL
    g0evt, g1evt, g2evt, g3evt : BOOL;
    g2d : BOOL;
END_VAR

(* Fim da seção de declaração de variáveis *)

(* Seção de código *)

IF (g0evt OR g1evt OR g2evt OR g3evt ) THEN
    g2d := TRUE;
ELSE
    g2d := FALSE;
END_IF;

(* Fim da seção de código *)

END_FUNCTION_BLOCK
```

Figura 5.7 – Function Block dg2

♦ fim do exemplo 5.3 ♦

5.5 SFCs e FBs no conjunto $\{s_j \mid j \in J\}$

Esta seção apresenta o procedimento sistemático para converter cada supervisor pertencente ao conjunto $\{\mathfrak{S}_j \mid j \in J\}$ no SFC correspondente. Tais SFCs constituem a seção de código dos FBs em $\{s_j \mid j \in J\}$. A *etapa-1* deste procedimento define as instâncias de tais FBs e as variáveis do CLP envolvidas nas respectivas seções de código. A *etapa-2* define o SFC s_j.

etapa-1) Definição de variáveis do CLP e instâncias dos FBs

Considere o conjunto de supervisores $\mathfrak{S}_j = (\mathbf{S_j}, \Phi_j)$, $j \in J$, com $\mathbf{S_j} = (\Sigma^{Sj}, Q^{Sj}, \delta^{Sj}, q0^{Sj}, Q_m{}^{Sj})$ e $\Phi_j : Q^{Sj} \rightarrow 2^{\Sigma c}$.

Em geral, há eventos em Σ_c que nunca são desabilitados por um supervisor em particular. Os eventos passíveis da ação de desabilitação do supervisor $\mathfrak{S}_j$ são aqueles em $\Sigma_p{}^{Sj} = \cup_{\forall q \in Q^{Sj}} \Phi_j(q)$.

Uma variável Booleana, denominada "σds_j" $\in P_j$, $j \in J$, é associada a cada evento $\sigma \in \Sigma_p{}^{Sj}$. O conjunto de variáveis P_j é tal que $|P_j| = |\Sigma_p{}^{Sj}|$. A variável σds_j é ativada (no FB s_j) sempre que o supervisor $\mathfrak{S}_j$ estabelece a desabilitação do evento associado a tal variável.

As variáveis associadas aos sinais de desabilitação (aquelas que pertencem aos conjuntos D_i, $i \in I$) são ativadas no FB MS com base no valor lógico das variáveis correspondentes nos conjuntos P_j, $j \in J$.

Cada FB s_j, $j \in J$, deve possuir exatamente uma instância denominada "Inst_s_j".

A Tabela 5.3 apresenta o escopo de definição das variáveis e FBs acima referidas nos diferentes POUs do programa de aplicação. Nenhuma destas variáveis precisa ser associada com variáveis de entrada ou saída do CLP.

Tabela 5.3 – Tipos de variáveis nos diferentes POUs

Instância de variáveis e FBs	Escopo de definição	POU
$\sigma ds_j \in Pj$ ($j \in J$)	*Global*	*Program* Main
	External	no FB s_j correspondente
		FB MS
Inst_s_j ($j \in J$)	*Global*	*Program* Main
	External	FB MS

♦ fim da etapa-1 ♦

etapa-2) Definição do SFC $s_j = (X^{sj}, T^{sj}, x0^{sj})$, $j \in J$

Define-se $\Sigma_{q,q'}{}^{Sj} = \{\sigma \in \Sigma \mid (q' = \delta^{Sj}(q, \sigma)) \wedge (q \neq q')\}$, o qual representa o conjunto de eventos cuja ocorrência pode causar a transição do estado q para o estado distinto q' do autômato $\mathbf{S_j}$ que representa o supervisor $\mathfrak{S}_j$.

O SFC s_j, $j \in J$, é obtido realizando os itens (*i*) a (*iii*) abaixo:

i) Um passo $xq \in X^{sj}$ é associado a cada estado $q \in Q^{Sj}$, de forma que $|X^{sj}| = |Q^{Sj}|$;

ii) $x0^{sj}$ é o passo associado ao estado inicial $q0^{Sj}$;

iii) Uma transição $(xq, xq') \in T^{sj}$ é associada a cada par ordenado $(q, q') \in \{Q^{Sj} \times Q^{Sj}\}$ sempre que $\Sigma_{q,q'}{}^{Sj} \neq \varnothing$, onde $xq, xq' \in X^{sj}$ são respectivamente associados à q, q'. Esta associação deve ser realizada de forma que $|T^{sj}|$ iguala o número de pares ordenados em que $\Sigma_{q,q'}{}^{Sj} \neq \varnothing$.

Este procedimento não estabelece prioridade à avaliação de transições no caso de seleção de seqüências divergentes.

A condição de transição associada à cada transição em T^{sj}, $j \in J$, é a expressão Booleana "(σ1 OR σ2 OR … OR σn)". Esta expressão assume que o conjunto de eventos que pode causar a transição do estado q para o estado q' do autômato $\mathbf{S_j}$ é $\Sigma_{q,q'}{}^{Sj} = \{\sigma 1, \sigma 2, \ldots, \sigma n\}$ onde (q, q') é o par

ordenado que originou tal transição. Ela também assume que os eventos $\sigma 1, \sigma 2, \ldots, \sigma n$ são, respectivamente, associados às variáveis $\sigma 1, \sigma 2, \ldots, \sigma n$ nos conjuntos E_i, $i \in I$.

As ações associadas ao passo $xq \in X^{sj}$ são na forma apresentada na Figura 5.8. Estas ações assumem que:

i) $xq \in X^{sj}$ é associado com $q \in Q^{Sj}$;

ii) $\Phi_j(q) = \{\sigma 1, \sigma 2, \ldots, \sigma n\}$;

iii) Os eventos $\sigma 1, \sigma 2, \ldots, \sigma n$ são, respectivamente, associados às variáveis $\sigma 1ds_j, \sigma 2ds_j, \ldots, \sigma nds_j$, onde $\{\sigma 1ds_j, \sigma 2ds_j, \ldots, \sigma nds_j\} \subseteq P_j$.

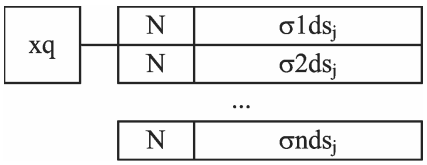


Figura 5.8 – Ações associadas ao passo xq do SFC s_j, $j \in J$

♦ fim da etapa-2 ♦

Exemplo 5.4)

Considere o supervisor $\mathfrak{S}_{c1}$ do exemplo motivador introduzido na Seção 3.3. Conforme apresentado na Tabela 3.2 a função de desabilitação de eventos deste supervisor é dada por: $\Phi_{c1}(0) = \{\alpha 2\}$, $\Phi_{c1}(1) = \{\alpha 1, \alpha 2\}$, $\Phi_{c1}(2) = \{\alpha 0\}$, $\Phi_{c1}(3) = \{\alpha 0, \alpha 1\}$. Com base nesta função define-se $\Sigma_p^{Sc1} = \Phi_{c1}(0) \cup \Phi_{c1}(1) \cup \Phi_{c1}(2) \cup \Phi_{c1}(3) = \{\alpha 0, \alpha 1, \alpha 2\}$ como o conjunto de eventos passíveis de desabilitação pelo supervisor $\mathfrak{S}_{c1}$. Conforme a *etapa-1* deste procedimento, realiza-se a associação "evento *x* variável do CLP" apresentada a seguir: ($\alpha 0$, a0dsc1), ($\alpha 1$, a1dsc1), ($\alpha 2$, a2dsc1). Esta associação resulta no conjunto de variáveis $P_{c1} = \{a0dsc1, a1dsc1, a2dsc1\}$.

Baseado no autômato $\mathbf{S_{c1}}$ apresentado na Figura 3.5 e conforme descrito na *etapa-2*, define-se $\Sigma_{0,1}^{Sc1} = \{\beta 1\}$, $\Sigma_{1,2}^{Sc1} = \{\alpha 0\}$, $\Sigma_{2,3}^{Sc1} = \{\beta 1\}$, $\Sigma_{2,0}^{Sc1} = \{\alpha 2\}$, $\Sigma_{3,1}^{Sc1} = \{\alpha 2\}$. Para todos os demais pares ordenados (q, q') em $\{Q^{Sc1} \times Q^{Sc1}\}$ tal que $q \neq q'$ tem-se que $\Sigma_{q,q'}^{Sc1} = \varnothing$.

A Figura 5.9 apresenta a seção de declaração de variáveis do FB sc1. Na Figura 5.10 é apresentado o SFC s_{c1}, o qual constitui a seção de código do referido FB. Este SFC é obtido realizando a associação " estado *x* passo" apresentada a seguir: (0, x0), (1, x1), (2, x2) e (3, x3).

```
FUNCTION_BLOCK sc1

(* Seção de declaração de variáveis *)

VAR_EXTERNAL
    a0, b1, b2 : BOOL;
    a0dsc1, a1dsc1, a2dsc1 : BOOL;
END_VAR

(* Fim da seção de declaração de variáveis *)
```

Figura 5.9 – Seção de declaração de variáveis do FB sc1

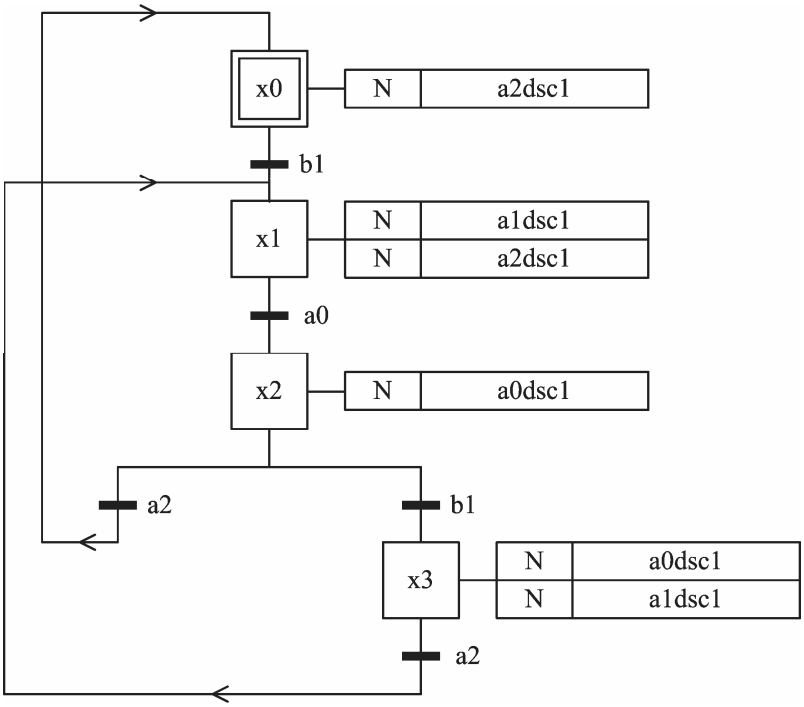


Figura 5.10 – Seção de código do FB sc1

♦ fim do exemplo 5.4 ♦

Exemplo 5.5)

Para ilustrar outro aspecto deste procedimento considere o supervisor $\mathfrak{S}_a$ (Figura 3.5). De acordo com a *etapa-2* define-se $\Sigma_{0,1}^{Sa} = \{\beta 1, \beta 2, \mu 2, \beta 3\}$ como o conjunto de eventos cuja ocorrência pode resultar na transição do estado 0 para o estado 1 do autômato $\mathbf{S_a}$. A condição de transição associada à transição (x0, x1) é a expressão Booleana (b1 OR b2 OR m2 OR b3). Esta expressão é baseada na associação "evento *x* variável do CLP" (β1, b1), (β2, b2), (μ2, m2), (β3, b3). O SFC s_a é apresentado na Figura 5.11 e assume a associação "estado do autômato $\mathbf{S_a}$ *x*

passo do SFC s_a" (0, x0), (1, x1), bem como a associação "evento *x* variável do CLP" (α0, a0) e (α0, a0dsa).

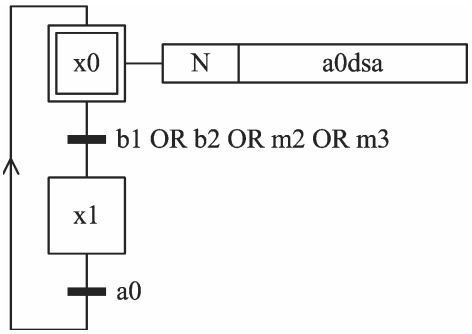


Figura 5.11 – SFC s_a

♦ fim do exemplo 5.5 ♦

5.6 Procedimentos operacionais no conjunto $\{o_\sigma \mid \sigma \in \Sigma\}$

Cada FB no conjunto $\{o_\sigma \mid \sigma \in \Sigma\}$ implementa o algoritmo de controle que dirige a execução das atividades pelo sistema a ser controlado. Em geral, a cada evento controlável em Σ_c é associado um procedimento operacional, isto associa cada comando a um procedimento operacional distinto. Porém, também é possível associar um procedimento operacional a um evento não-controlável.

A evolução do SFC o_σ, $\sigma \in \Sigma$, é determinada pelo estado lógico do comando associado a tal procedimento operacional bem como pelo estado dos sinais dos sensores instalados no sistema a ser controlado. Este comando, e os sinais dos sensores, são empregados para estabelecer o valor lógico das condições de transição do referido SFC. As ações associadas aos passos deste SFC estabelecem os sinais que devem ser enviados aos atuadores instalados no sistema a ser controlado. A execução das atividades pode resultar na ocorrência de um evento não-controlável.

Conforme mencionado na Seção 5.2, considera-se que a obtenção de um SFC que representa um procedimento operacional é parte do processo de modelagem do sistema. Como orientações genéricas à modelagem destes SFCs recomenda-se que:

i) Se o procedimento operacional está associado a um evento controlável, e portanto a um comando, tal comando deve ser empregado para compor as expressões Booleanas que representam a condição de transição associada a toda transição cujo passo predecessor é o passo inicial do SFC. Além disto, este comando deve ser desativado no passo sucessor à transição cuja condição de

transição considera o referido comando. A desativação do comando confirma que o mesmo já foi processado pelo procedimento operacional correspondente;

ii) Se o procedimento operacional é o responsável por identificar a ocorrência de um determinado evento não-controlável, então, na ativação do passo do SFC em que é identificada a ocorrência de tal evento a resposta associada ao evento em questão deve ser incrementada de uma unidade;

iii) A evolução do SFC que implementa o procedimento operacional resulta, em algum momento, na ativação do passo inicial.

Exemplo 5.6)

Considere o sistema de manufatura apresentado na Seção 3.3, onde $\Sigma_c = \{\alpha 0, \alpha 1, \tau 1, \alpha 2, \alpha 3, \tau 3, \alpha 4, \alpha 5\}$. A cada evento em Σ_c foi associado um procedimento operacional, de forma que O = {oa0, oa1, ot1, oa2, oa3, ot3, oa4, oa5} é o conjunto de procedimentos operacionais.

A Figura 5.12 apresenta uma representação esquemática do Sistema de furação e a Figura 5.13 o SFC oa2 que representa o procedimento operacional associado ao evento $\alpha 2$ e ao comando cmda2. Neste sistema o atuador E realiza o avanço e recuo linear da furadeira e o atuador G realiza a fixação da peça na posição de furação. O acionamento rotativo da furadeira é realizado pelo motor elétrico identificado como driller. O sensor drill-ok retorna valor Verdadeiro sempre que a broca está íntegra. O indicador luminoso alrm2 sinaliza que o sistema está no estado "em pane".

Seguindo a orientação (*i*) apresentada acima, o comando cmda2 é empregado nas expressões Booleanas associadas à condição de transição das transições (x0, x1) e (x0, x6). Este comando é desativado em ações associadas aos passos x1 e x6. A geração do evento $\alpha 2$ (iniciar furação) e a ativação do comando cmda2 disparam a execução das atividades deste subsistema. A desativação do comando cmda2 nos passos x1 ou x6 confirma que este comando já foi processado pelo procedimento operacional. A execução destas atividades resulta na ocorrência do evento $\beta 2$ (furado com sucesso) ou do evento $\lambda 2$ (pane na furação).

O evento $\beta 2$ pode ocorrer quando o sistema está no estado representado pelo passo x4. A ocorrência deste evento implica que a expressão Booleana “G_bwd AND drill_ok” assume valor lógico VERDADEIRO. Esta é a condição de transição associada à transição (x4, x5). O evento $\lambda 2$ pode ocorrer quando o sistema está nos estados representados pelo passo inicial (x0) e pelo passo x4. A ocorrência deste evento implica que a expressão Booleana “cmda2 AND NOT drill_ok” assume valor lógico VERDADEIRO, ou que a expressão Booleana “G_bwd AND NOT drill_ok” assume tal valor lógico. A primeira expressão é a condição de transição associada à transição (x0, x6), e a segunda é a condição de transição associada à transição (x4, x6). Mesmo que a ocorrência

dos eventos não tenha duração no tempo, o valor lógico destas expressões irá durar por um tempo significativo, resultando na ativação do passo x5 ou do passo x6.

Seguindo a orientação (*ii*) apresentada acima, a ativação do passo x5, devida à ocorrência do evento β2, resulta no incremento de uma unidade da resposta rspb2. Visto que esta resposta é incrementada apenas na ativação do passo x5, uma ocorrência do evento β2 será tratada exatamente uma vez, independentemente do tempo em que tais expressões Booleanas permaneçam no valor lógico VERDADEIRO.

A evolução do SFC com a ativação dos passos x5 ou x7 resulta na ativação do passo inicial (orientação (*iii*)).

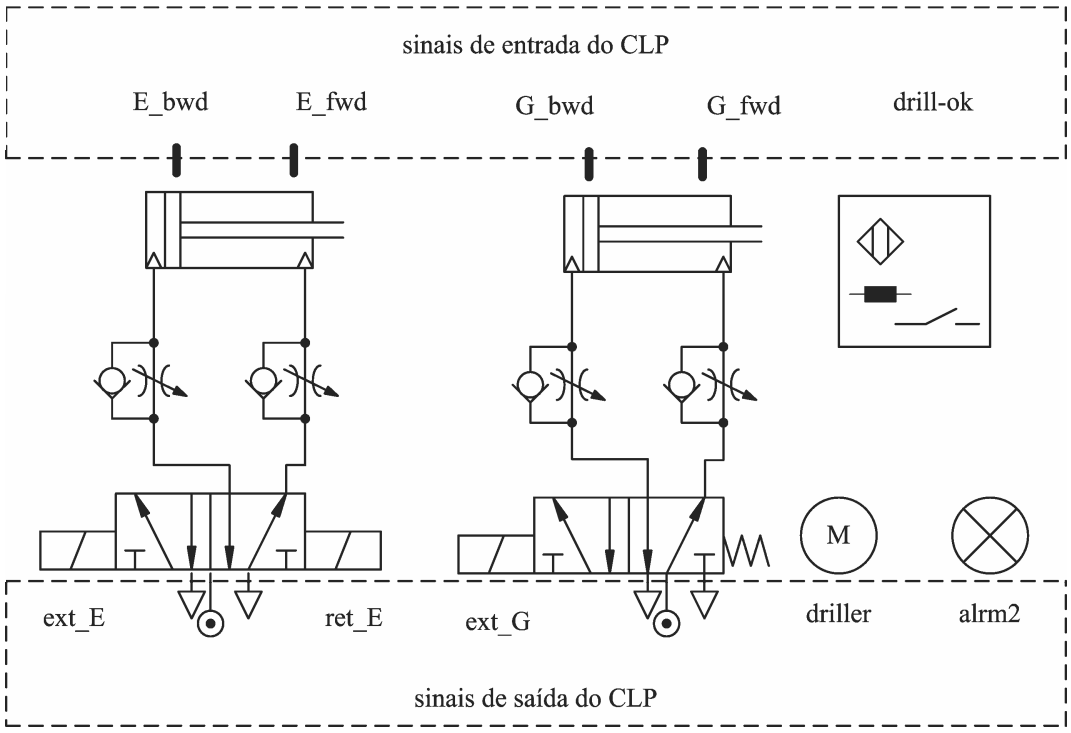


Figura 5.12 – Representação esquemática do sistema de furação

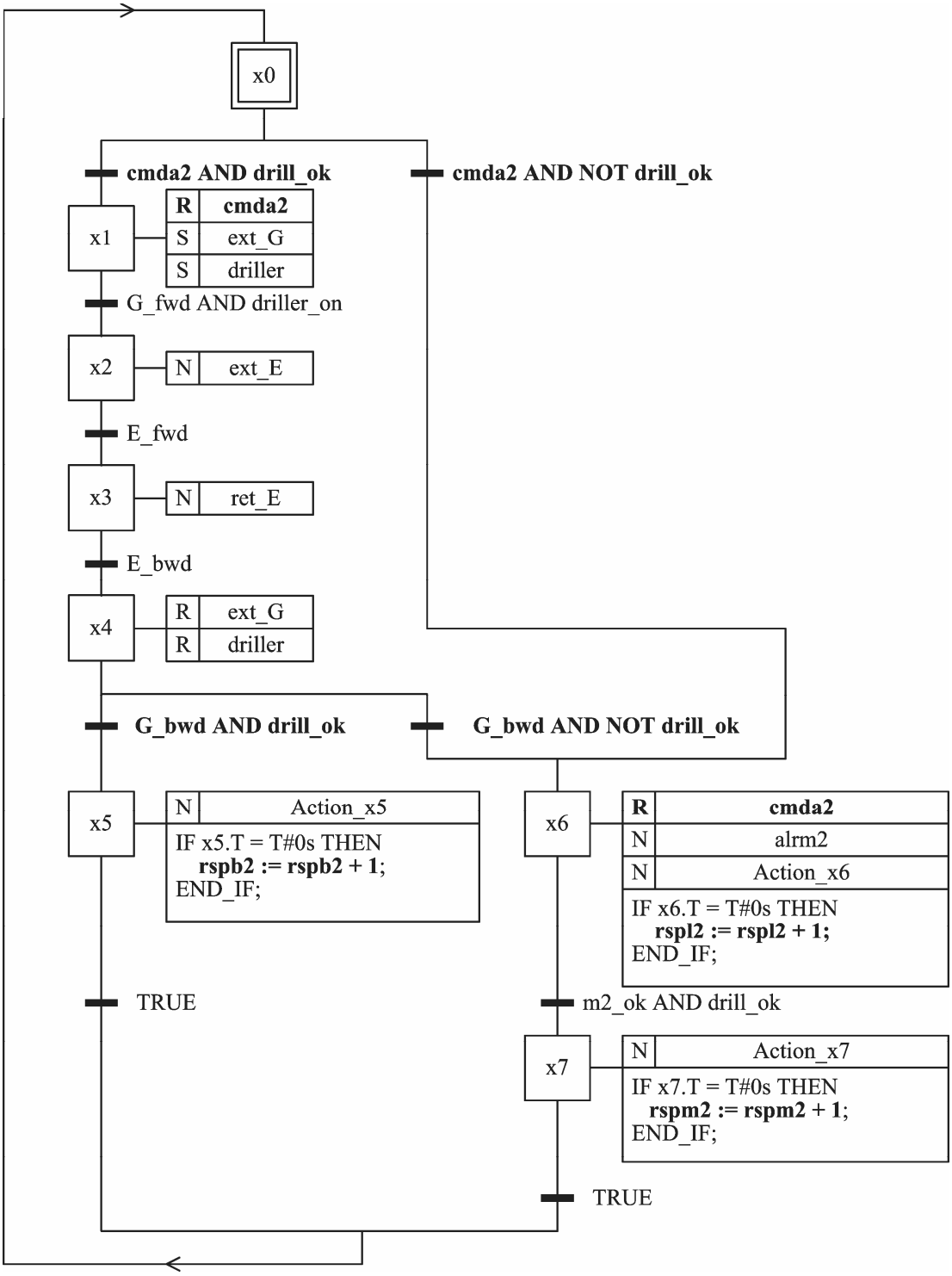


Figura 5.13 – SFC oa2

♦ fim do exemplo 5.6 ♦

Exemplo 5.7)

Considerou-se que, devido às características apresentadas pela célula de manufatura apresentada na Seção 3.3, a decisão de onde armazenar cada peça pode ser estabelecida de forma

trivial, pois a ordem em que as peças chegam ao manipulador robótico é a ordem na qual as peças deixam o sistema de classificação e transporte. Assim, o sistema de classificação e transporte alimenta uma estrutura FIFO (*First In First Out*) com o tipo de peça que deixa este subsistema. O manipulador robótico, no momento de realizar a armazenagem, resgata informações desta estrutura para reconhecer qual o tipo de peça a ser armazenada. O mesmo se aplica com o resultado do teste. O sistema de teste alimenta uma segunda estrutura FIFO com o resultado do teste, o qual é resgatado pelo manipulador robótico no momento de armazenar a peça.

Tais soluções de controle foram implementadas nos procedimentos operacionais oa1, oa3, ot3 e oa4, respectivamente associados aos eventos $\alpha 1$, $\alpha 3$, $\tau 3$ e $\alpha 4$. O procedimento operacional oa1 alimenta a primeira estrutura FIFO com o tipo de peça. Os procedimentos operacionais oa3 e ot3 alimentam a segunda estrutura com o resultado do teste. O procedimento operacional oa4 resgata os dados destas duas estruturas.

♦ fim do exemplo 5.7 ♦

5.7 FBs MS, PS e OP

O SFC Main gerencia a chamada aos FBs nos conjuntos $\{s_j \mid j \in J\}$, $\{g_i \mid i \in I\}$, $\{dg_i \mid i \in I\}$, e $\{o_\sigma \mid \sigma \in \Sigma\}$. Isto é realizado com a ajuda dos FBs denominados MS, PS e OP.

O FB MS deve realizar a chamada seqüencial a todos os FBs no conjunto $\{s_j \mid j \in J\}$. A chamada a um FB em $\{s_j \mid j \in J\}$ atualiza o estado ativo do supervisor com os eventos tratados no ciclo de atualização anterior. A ordem em que os FBs $\{s_j \mid j \in J\}$ são chamados é irrelevante.

O FB MS também deve estabelecer o estado dos sinais de desabilitação, ou seja, o estado das variáveis nos conjuntos D_i, $i \in I$. Considere a variável $\sigma d \in D_i$, $i \in I$, e todas as variáveis nos conjuntos P_j, $j \in J$, que estão associadas ao mesmo evento que a variável em questão (ver *etapa-1* na Seção 5.5), à esta variável σd deve ser estabelecido o valor lógico VERDADEIRO sempre que pelo menos uma das variáveis correspondentes nos conjuntos P_j assumir valor lógico VERDADEIRO. Quando todas as variáveis correspondentes nos conjuntos P_j assumirem valor lógico FALSO, então, à variável σd deve ser estabelecido o valor lógico FALSO.

Exemplo 5.8)

Considerando o exemplo introduzido na Seção 3.3, a Tabela 5.4 apresenta os conjuntos de eventos passíveis de desabilitação pela ação de controle dos diversos supervisores. Desta tabela, verifica-se que a ocorrência do evento $\alpha 2$ é passível de desabilitação pelos supervisores $\mathfrak{S}_{b2}$, $\mathfrak{S}_{c1}$ e $\mathfrak{S}_{c2}$. A Figura 5.14 apresenta como o estado da variável $a2d \in D_2$, associada ao evento $\alpha 2$, é

determinado no FB MS pela ação do conjunto de supervisores. Assume-se que as variáveis a2dsb2 $\in P_{b2}$, a2dsc1 $\in P_{c1}$ e a2dsc2 $\in P_{c2}$ também estão associadas ao evento $\alpha 2$.

Tabela 5.4 – Eventos passíveis de desabilitação pelos supervisores	
conjunto de eventos	conjunto de variáveis do CLP
$\Sigma_p^{Sa} = \{\alpha 0\}$	$P_a = \{a0dsa\}$
$\Sigma_p^{Sb1} = \{\alpha 0, \alpha 1\}$	$P_{b1} = \{a0dsb1, a1dsb1\}$
$\Sigma_p^{Sb2} = \{\alpha 0, \alpha 2\}$	$P_{b2} = \{a0dsb2, a2dsb2\}$
$\Sigma_p^{Sb3} = \{\alpha 0, \alpha 3\}$	$P_{b3} = \{a0dsb3, a3dsb3\}$
$\Sigma_p^{Sb4} = \{\alpha 0, \alpha 4\}$	$P_{b4} = \{a0dsb4, a4sb4\}$
$\Sigma_p^{Sc1} = \{\alpha 0, \alpha 1, \alpha 2\}$	$P_{c1} = \{a0dsc1, a1dsc1, a2dsc1\}$
$\Sigma_p^{Sc2} = \{\alpha 0, \alpha 2, \alpha 3, \tau 3\}$	$P_{c2} = \{a0dsc2, a2dsc2, a3dsc2, t3dsc2\}$
$\Sigma_p^{Sc3} = \{\alpha 0, \alpha 3, \tau 3, \alpha 4\}$	$P_{c3} = \{a0dsc3, a3dsc3, t3dsc3, a4dsc3\}$
$\Sigma_p^{Sd} = \{\tau 1, \alpha 5\};$	$P_d = \{t1dsd, a5dsd\}$

```
IF (a2dsb2 OR a2dsc1 OR a2dsc2) THEN
    a2d := TRUE;
ELSE
    a2d := FALSE;
END_IF;
```

Figura 5.14 – Determinação do estado da variável a2d no FB MS

♦ fim do exemplo 5.8 ♦

O FB PS deve realizar a chamada seqüencial de todos os FBs nos conjuntos $\{dg_i \mid i \in I\}$ e $\{g_i \mid i \in I\}$. Imediatamente antes de chamar o FB g_i, deve ser chamado o FB dg_i correspondente. O FB PS deve chamar o FB g_i somente se o tratamento de eventos em Σ^{Gi} não foi suspenso pelo FB dg_i, isto é, somente se a variável g_id apresentar valor lógico FALSO. Chamar o FB g_i pode resultar no tratamento de evento (geração de evento controlável ou sinalização da ocorrência de evento não-controlável) em Σ^{Gi}.

Apresentou-se na Seção 4.2.3 que, priorizar o tratamento de um determinado evento em detrimento dos demais, pode resultar no bloqueio do comportamento do sistema sob ação de supervisão. A ordem em que é realizada a chamada aos FBs em $\{g_i \mid i \in I\}$ especifica esta priorização no tratamento de eventos. O primeiro FB a ser chamado em $\{g_i \mid i \in I\}$ tem a maior prioridade de tratamento de eventos. Caso o conjunto de propriedades definidas em (DIETRICH *et*

al., 2002) seja satisfeito, então, a ordem em que é realizada a chamada a estes FBs é irrelevante. Em caso contrário, o problema do bloqueio pode se manifestar.

O FB OP deve realizar a chamada seqüencial a todos os FBs que implementam procedimentos operacionais. A chamada aos FBs em $\{o_\sigma \mid \sigma \in \Sigma\}$ atualiza o algoritmo de controle que dirige a execução das atividades pelo sistema a ser controlado. A ordem em que os FBs em $\{o_\sigma \mid \sigma \in \Sigma\}$ são chamados é irrelevante.

5.8 Ações associadas aos passos do SFC Main

O SFC Main coordena a operação da arquitetura de controle, sendo que o passo ativo deste SFC determina o modo de operação do sistema. A ação associada a cada passo deste SFC determina o comportamento do sistema em cada um dos diferentes modos de operação.

Ação action_SI

Quando o passo SI do SFC Main é ativado a ação action_SI é executada apenas uma vez. Esta ação deve estabelecer o valor lógico FALSO a todas as variáveis em $\{E_i \mid i \in I\}$, $\{C_i \mid i \in I\}$, $\{D_i \mid i \in I\}$, $\{g_i evt \mid i \in I\}$, $\{g_i d \mid i \in I\}$, $\{\sigma ds_j \mid \sigma \in \Sigma_p^{Sj} \wedge j \in J\}$, também deve atribuir às variáveis em $\{R_i \mid i \in I\}$ o valor zero. À variável safe deve ser atribuído o valor lógico VERDADEIRO. Esta ação deve inicializar todos os SFCs nos conjuntos $\{s_j \mid j \in J\}$, $\{g_i \mid i \in I\}$ e $\{o_\sigma \mid \sigma \in \Sigma\}$.

Ação action_Sup

Quando o passo Sup do SFC Main é ativado, e enquanto este passo permanece ativo, o sistema está no modo *Supervised* e a ação action_Sup é executada uma vez a cada ciclo de atualização do CLP. Esta ação executa a seguinte seqüência de atividades:

i–a) Atualização do estado ativo dos supervisores com os eventos tratados no ciclo de atualização anterior, bem como, atualização da ação de controle dos supervisores. Isto é realizado através da chamada ao FB MS;

ii) Desativação condicional das variáveis em $\{g_i evt \mid i \in I\}$. À cada variável em $\{g_i evt \mid i \in I\}$ deve ser estabelecido o valor lógico FALSO sempre que toda variável no conjunto C_i correspondente tiver valor lógico FALSO (ver Exemplo 5.9);

iii) Suspensão do tratamento de todo evento controlável para priorizar o tratamento de eventos não-controláveis. Isto é realizado através da ativação da variável denominada CED;

iv) Tratamento de eventos não-controláveis. Se ocorreu algum evento não-controlável e a resposta correspondente armazena valor maior do que zero, então a ocorrência do evento será sinalizada. Isto é realizado através da chamada ao FB PS;

v) Autorização do tratamento de eventos controláveis. Isto é realizado através da desativação da variável mencionada no ítem (*iii*);

vi) Tratamento de eventos controláveis, realizado através de nova chamada ao FB PS;

vii) Atualização de todos os procedimentos operacionais em $\{o_\sigma \mid \sigma \in \Sigma\}$. Isto é realizado através da chamada ao FB OP.

Exemplo 5.9)

Considere o subsistema $\mathbf{G_1}$ do exemplo motivador introduzido na Seção 3.3, onde $\Sigma_c^{G1} = \{\alpha 1, \tau 1\}$. Considere que tenha sido realizada a associação "evento *x* variável do CLP" $(\alpha 1, \text{cmda1})$ e $(\tau 1, \text{cmdt1})$, com $C_1 = \{\text{cmda1}, \text{cmdt1}\}$, bem como a associação "subsistema *x* variável do CLP" ($\mathbf{G_1}$, g1evt). A Figura 5.15 apresenta como é realizada a desativação condicional da variável g1evt em função do especificado no item (*ii*) acima.

```
IF  (NOT cmda1 AND NOT cmdt1) THEN
  g1evt := FALSE;
END_IF;
```

Figura 5.15 – Desativação condicional da variável g1evt

♦ fim do exemplo 5.9 ♦

Ação action_Man

Quando o passo Man do SFC Main é ativado, e enquanto este passo permanece ativo, o sistema está no modo *Manual* e a ação action_Man é executada uma vez a cada ciclo de atualização do CLP. A seqüência de atividades executada nesta ação é muito similar àquela executada pela ação action_Sup, porém, imediatamente após executar o item (*i–a*) os itens (*i–b*) a (*i–e*) devem ser executados antes do item (*ii*):

i–b) Determinação e sinalização ao operador se, no ciclo de atualização anterior, houve a geração de algum evento controlável que estaria desabilitado pelo conjunto de supervisores. Em caso afirmativo significa que alguma especificação de controle e o comportamento ótimo sob supervisão foram violados;

i–c) Memorização e sinalização ao operador da ação de controle atual do conjunto de supervisores (eventos controláveis que devem ser desabilitados para preservar o comportamento ótimo sob

supervisão). Esta informação será utilizada para realizar o item (*i*–*b*) no próximo ciclo de atualização do CLP;

i–*d*) Desabilitação de todos os eventos controláveis. Isto é realizado através da ativação de todas as variáveis nos conjuntos D_i, $i \in I$;

i–*e*) Habilitação de eventos controláveis em função da ação do operador. Isto é realizado pela desativação seletiva de variáveis nos conjuntos D_i, $i \in I$.

Destaca-se que neste modo de operação o operador é o responsável pela coordenação dos diversos subsistemas. Sua ação de controle consiste na habilitação seletiva de eventos controláveis. Como auxílio na tomada de decisão o operador pode consultar a ação de controle do conjunto de supervisores, contudo, não é necessário acatá-la.

Destaca-se ainda que a habilitação de um evento controlável em Σ_c^{Gi} não determina sua ocorrência, é necessário também que, após a seqüência de eventos já tratada, a ocorrência deste evento seja prevista pelo modelo que descreve o comportamento independente do subsistema $\mathbf{G_i}$. Além disto, o tratamento de eventos não-controláveis sempre tem prioridade sobre o tratamento de eventos controláveis.

Para implementar os itens *i*–b a *i*–e, a cada evento controlável $\sigma \in \Sigma_c^{Gi}$ devem ser associadas duas variáveis Booleanas designadas σdprev e Enableσ. A primeira variável é empregada para memorizar (no item *i*–c) o estado do sinal de desabilitação correspondente no ciclo de atualização atual. O valor armazenado nesta variável é utilizado para determinar (no item *i*–b) se houve a geração de algum evento que estava desabilitado pela ação de controle dos supervisores. A segunda variável é empregada para o operador estabelecer a habilitação do evento correspondente. Tais variáveis tem escopo de definição *Global*. Em geral o estado de tais variáveis é estabelecido através de uma interface com o usuário desenvolvida em um sistema de supervisão.

Exemplo 5.10)

A ação action_Man para o exemplo introduzido na Seção 3.3 foi implementada conforme apresentado nas Figuras 5.16 e 5.17. Nestas figuras inst_MS, inst_PS e Inst_OP são, respectivamente, as instâncias dos FBs MS, PS e OP. Estes FBs possuem uma variável Booleana de entrada denominada init. Na ação action_SI estes FBs são chamados com o valor VERDADEIRO associado a esta variável, isto inicializa os SFCs em $\{s_j \mid j \in J\}$, $\{g_i \mid i \in I\}$ e $\{o_\sigma \mid \sigma \in \Sigma\}$. Nas ações action_Sup, action_Man e action_PSI estes FBs são chamados com o valor FALSO associado a esta variável.

Na Figura 5.16 R_TRIG_a0 é uma instância do FB R_TRIG. Este FB retorna o valor lógico VERDADEIRO à variável de saída (R_TRIG_a0.Q) quando é identificada a transição

positiva de sinal da variável de entrada do FB (CLK). O estado da variável Enable_a0 é determinado pelo operador. Para habilitar a ocorrência do evento a0 o operador deve estabelecer o valor VERDADEIRO a esta variável.

A ação action_SUP é praticamente idêntica à ação action_Man. Ela é obtida excluindo os itens (*i-b*) até (*i-e*) apresentados nas Figuras 5.16 e 5.17

```
 (* action_Man: itens i-a até i-e *)

(* item i-a *)
inst_MS( init := FALSE);

(* item i-b *)
IF ( (a0 AND a0dprev) OR (a1 AND a1dprev) OR (t1 AND t1dprev) OR (a2 AND a2dprev) OR
        (a3 AND a3dprev) OR (t3 AND t3dprev) OR (a4 AND a4dprev) OR (a5 AND a5dprev) )
THEN
   safe := FALSE;
END_IF;

(* item i-c *)
a0dprev := a0d;                                 a1dprev := a1d;
t1dprev := t1d;                                 a2dprev := a2d;
a3dprev := a3d;                                 t3dprev := t3d;
a4dprev := a4d;                                 a5dprev := a5d;

(* item i-d *)
a0d := TRUE;                                    a1d := TRUE;
t1d := TRUE;                                    a2d := TRUE;
a3d := TRUE;                                    t3d := TRUE;
a4d := TRUE;                                    a5d := TRUE;

(* item i-e *)
R_TRIG_a0(CLK := Enablea0);
IF R_TRIG_a0.Q          THEN        a0d := FALSE;           END_IF;
R_TRIG_a1(CLK := Enablea1);
IF R_TRIG_a1.Q          THEN        a1d := FALSE;           END_IF;
R_TRIG_t1(CLK := Enablet1);
IF R_TRIG_t1.Q          THEN        t1d := FALSE;           END_IF;
R_TRIG_a2(CLK := Enablea2);
IF R_TRIG_a2.Q          THEN        a2d := FALSE;           END_IF;
R_TRIG_a3(CLK := Enablea3);
IF R_TRIG_a3.Q          THEN        a3d := FALSE;           END_IF;
R_TRIG_t3(CLK := Enablet3);
IF R_TRIG_t3.Q          THEN        t3d := FALSE;           END_IF;
R_TRIG_a4(CLK := Enablea4);
IF R_TRIG_a4.Q          THEN        a4d := FALSE;           END_IF;
R_TRIG_a5(CLK := Enablea5);
IF R_TRIG_a5.Q          THEN        a5d := FALSE;           END_IF;
```

Figura 5.16 – Ação action_Man, itens (*i-a*) até (*i-e*)

```
(*action_Man: itens ii até vi *)

(* item ii *)
IF  (NOT cmda0)           THEN          g0evt := FALSE;          END_IF;
IF  (NOT cmda1 AND NOT cmdt1)           THEN          g1evt := FALSE;          END_IF;
IF  (NOT cmda2)           THEN          g2evt := FALSE;          END_IF;
IF  (NOT cmda3 AND NOT cmdt3)           THEN          g3evt := FALSE;          END_IF;
IF  (NOT cmda4)           THEN          g4evt := FALSE;          END_IF;
IF  (NOT cmda5)           THEN          g5evt := FALSE;          END_IF;

(* item iii *)
CED := TRUE;

(* item iv *)
inst_PS (init := FALSE);

(* item v *)
CED := FALSE;

(* item v *)
inst_PS(init := FALSE);

(* item vi *)
inst_OP(init := FALSE);
```

Figura 5.17 – Ação action_Man, itens (*ii*) até (*vi*)

♦ fim do exemplo 5.10 ♦

Ação action_PSI

Enquanto o passo PSI do SFC Main está ativo o sistema está no modo *Physical System Initialization* e a ação action_PSI é executada uma vez a cada ciclo de atualização do CLP.

Considera-se que a condução do sistema a ser controlado para o estado inicial pode ser implementada através de um processo seqüencial. Uma possível forma de representar este processo é através de um SFC. Visto que algumas das atividades a serem executadas neste processo correspondem aos procedimentos operacionais conforme descritos na Seção 5.6, este SFC pode ser implementado de forma a ativar adequadamente comandos (variáveis nos conjuntos C_i, $i \in I$) na ativação de cada um dos passos e continuamente realizar a chamada ao FB OP para executar tais procedimentos. As condições de transição deste SFC podem ser associadas às respostas (variáveis nos conjuntos R_i, $i \in I$).

Caso hajam atividades a serem executadas neste processo que não possam ser executadas por quaisquer dos procedimentos operacionais associados aos eventos em Σ, podem ser, então, elaborados procedimentos operacionais específicos para este fim. A tais procedimentos são associados comandos e respostas de inicialização do sistema. O conjunto de procedimentos operacionais é então expandido com a inclusão dos procedimentos de inicialização.

Exemplo 5.11)

Para realizar o processo que conduz a célula de manufatura ao estado inicial, foi elaborado um conjunto de procedimentos de inicialização. A Tabela 5.5 relaciona os diversos subsistemas, os procedimentos de inicialização associados, e os respectivos comandos e respostas.

Tabela 5.5 – Procedimentos de inicialização

subsistema	procedimento	comando	resposta
M_0	-	-	-
M_1	iniG1A	iniG1Astart	iniG1Aend
	iniG1B	iniG1Bstart	iniG1Bend
M_2	iniG2	iniG2start	iniG2end
M_3	iniG3	iniG3start	iniG3end
M_4	-	-	-
M_5	iniG5A	iniG5Astart	iniG5Aend
	iniG5B	iniG5Bstart	iniG5Bend

Desta forma o conjunto de procedimentos operacionais associados a este sistema é dado por {oa0, oa1, ot1, oa2, oa3, ot3, oa4, oa5} $\cup$ {iniG1A, iniG1B, iniG2, iniG3, iniG5A, iniG5B}.

O SFC apresentado na Figura 5.18 implementa a ação action_PSI.

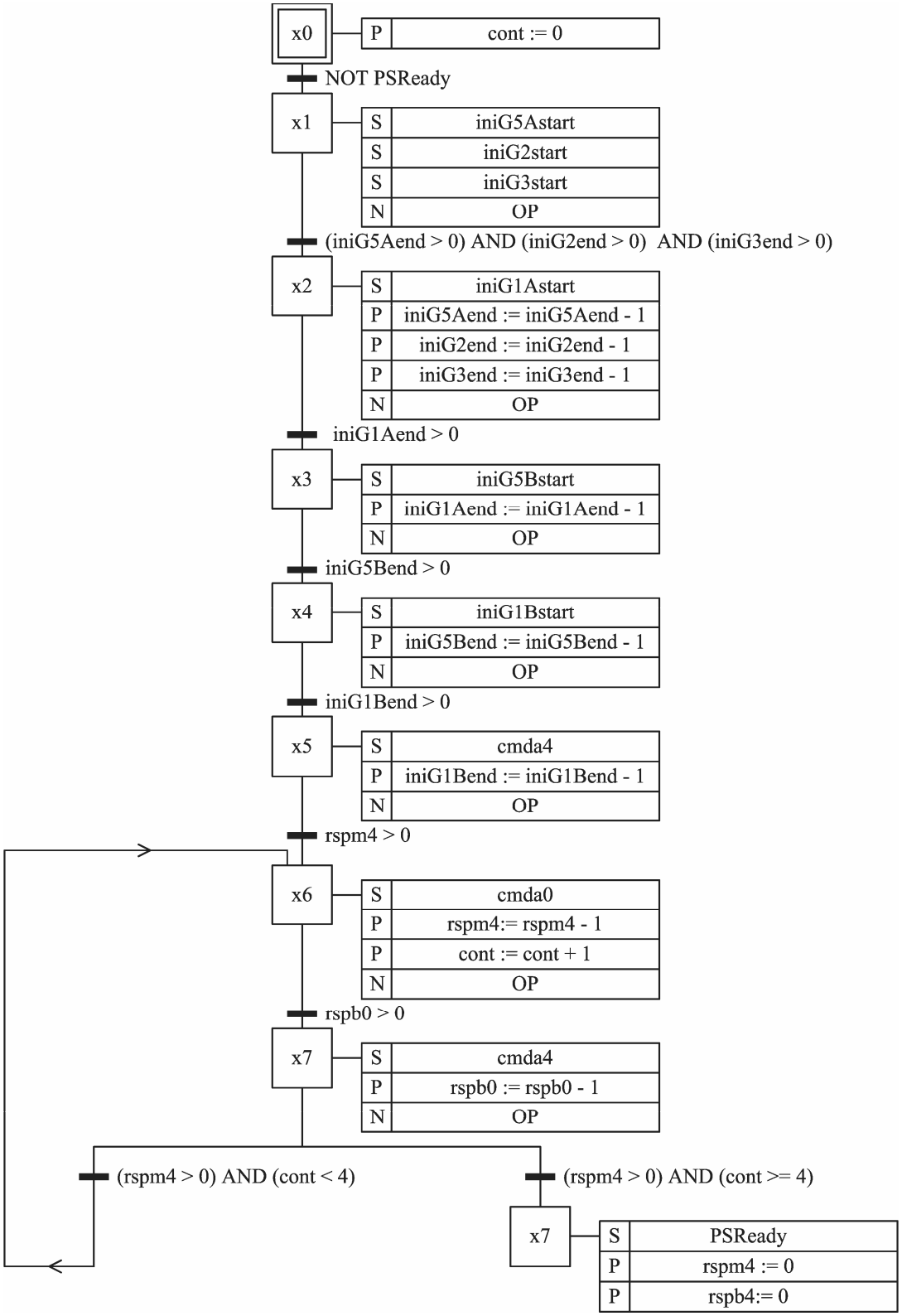


Figura 5.18 – Ação action_PSI

♦ fim do exemplo 5.11 ♦

Ação action_Emg

Em geral, no caso de uma situação de emergência todas as ações são imediatamente suspensas e alguns atuadores e alarmes devem ser devidamente ativados enquanto os demais são desativados. Esta ação pode ser implementada estabelecendo adequadamente o valor das variáveis de saída do CLP.

5.9 Discussão sobre o método

O método de implementação definido neste capítulo é um procedimento sistemático que, baseado no modelo que descreve o comportamento livre do sistema a ser controlado e nos supervisores sintetizados em conformidade com a Teoria de Controle Supervisório, define uma grande parte do código a ser implementado no CLP. Considera-se que o desenvolvimento dos procedimentos operacionais, a parcela de código que não é definida pelo método de implementação, faz parte da etapa de modelagem do sistema.

O código resultante da aplicação do método de implementação é estruturado em diversos *Function Blocks* e um *Program*. Visto que cada um destes POUs está diretamente relacionado com algum elemento em $\{\mathfrak{F}_j \mid j \in J\}$, $\{\mathbf{G_i} \mid i \in I\}$ ou $\{\mathbf{o_\sigma} \mid \sigma \in \Sigma\}$, a adoção do método de implementação resulta em um programa de aplicação com as seguintes características:

i) facilidade de interpretação do código;

ii) facilidade de avaliação e correção do código;

iii) facilidade de alteração do código em caso de:

– inclusão ou exclusão de subsistemas;

– alteração das especificações de controle;

– alteração da solução tecnológica adotada para realizar as atividades e tarefas;

iv) facilidade de reaproveitamento de código para outras implementações.

Destaca-se que alguns ambientes de programação de CLPs não oferecem suporte à linguagem SFC. Em tal caso a estrutura representada pelo SFC (arranjo de passos e transições) pode ser facilmente implementada em qualquer outra linguagem de programação. HELLGREN *et al.* (2005), bem como BOLLMANN (1996) apresentam como realizar a representação de um SFC na linguagem *Ladder Diagram*. Além disto, este método pode ser facilmente adaptado para outros tipos de controladores industriais, tais como, microcontroladores e computadores de uso geral.

É possível desenvolver compiladores que, baseados na representação por sistema produto e no conjunto de supervisores, realizem a geração automática de código para um modelo de CLP específico.

Este método de implementação foi aplicado com sucesso para implementar o controle da célula de manufatura introduzida na Seção 3.3. O sistema de controle utilizado foi um CLP da marca SIEMENS modelo 317-2 DP. Este método de implementação foi avaliado com sucesso para o controle de um sistema mais simples (A pequena fábrica, introduzido em (CURY, 2001)) em um SoftPLC comercial denominado Isagraf da ICS Triplex (ISAGRAF, 2007). Visto que a versão de demonstração não oferece suporte à comunicação do sistema de controle com o sistema a ser controlado tal avaliação foi realizada apenas no modo de simulação.

O método de implementação também foi empregado com sucesso no controle do sistema discutibo em (SILVA *et al.*, 2007a, 2007b).

A Arquitetura de Controle Supervisório proposta por Queiroz e Cury, juntamente com o método de implementação, realiza o tratamento dos aspectos abordados na Seção 4.2. Na seqüência é apresentado como é realizado o tratamento de cada um destes aspectos.

Causalidade

Da Figura 4.13 verifica-se que os níveis Sistema Produto, Procedimentos Operacionais e Sistema a ser controlado constituem a Planta Ramadge e Wonham. Desta forma, sob o ponto de vista dos supervisores, todos os eventos são gerados pela planta, o que preserva a hipótese adotada pela Teoria de Controle Supervisório (RAMADGE e WONHAM, 1987). Assim, os supervisores preservam a função original, a qual é a habilitação ou desabilitação de eventos controláveis. Este fato permite o livre emprego de representações reduzidas dos supervisores, o que não é válido quando os supervisores exercem o papel de gerar uma parcela de eventos. A implementação da representação reduzida dos supervisores resulta na alocação racional de memória do CLP, o que pode ser crucial na viabilização da implementação. Além disto, a interpretação, correção e/ou alteração do código é tanto mais fácil quanto menor o número de estados empregado na representação de cada um destes supervisores.

Exemplo 5.12)

No exemplo motivador apresentado na Seção 3.3, a adoção da abordagem modular para síntese de supervisores resultou em um conjunto de nove supervisores. Na Tabela 5.6 é apresentado o número total de estados e transições do conjunto de supervisores modulares, sob representação reduzida e não-reduzida. Estas informações também são apresentadas para a abordagem monolítica para síntese do supervisor.

Tabela 5.6 – N° de estados e N° de transições do conjunto de supervisores			
Abordagem monolítica		Abordagem modular local	
representação não-reduzida	representação reduzida	representação não-reduzida	representação reduzida
2.082 estados 6.914 transições	362 estados 2.442 transições	302 estados 994 transições	29 estados 68 transições

Sabendo que a memória de CLP necessária para realizar a implementação é proporcional ao número de estados do conjunto de supervisores, verifica-se da Tabela 5.6 que a redução dos supervisores é um aspecto fundamental na viabilização da implementação.

♦ fim do exemplo 5.12 ♦

Sinais e eventos

No método de implementação a ocorrência de eventos é representada por variáveis distintas daquelas que representam os sinais trocados entre o sistema de controle e o sistema a ser controlado. A geração de um evento controlável resulta na ativação, durante um ciclo de atualização do CLP, de uma variável interna ao CLP. Tal variável é empregada para atualizar o estado ativo do conjunto de supervisores. A execução das atividades associadas a este evento é realizada em um procedimento operacional. Por outro lado, a ocorrência de um evento não-controlável é detectada por uma combinação lógica de variáveis de entrada e variáveis internas do CLP. A ocorrência deste evento é sinalizada aos supervisores através da ativação de uma variável específica para este fim. A adequada modelagem e implementação dos procedimentos operacionais garante que os problemas denominados perda de informação e duração dos comandos não se manifestam.

Em função das regras de evolução de SFCs apresentadas em (LEWIS, 1998), um passo de um SFC não pode ser ativado e desativado na mesma chamada do FB cuja seção de código é constituída pelo SFC em questão. Como cada estado dos supervisores é implementado por um passo do SFC associado ao supervisor em questão, então o efeito avalanche não se manifesta.

Para que o problema associado à incapacidade de reconhecer a ordem de eventos não se manifeste, é fundamental que o comportamento ótimo sob supervisão obtido pela ação conjunta dos supervisores modulares seja insensível ao entrelaçamento com relação à planta **G**.

Escolha

Conforme mencionado na Seção 5.6, a ordem em que os FBs em $\{g_i \mid i \in I\}$ são chamados no FB PS determina a prioridade de tratamento de eventos em Σ. O problema de bloqueio devido à

escolha pode se manifestar se as propriedades definidas em (DIETRICH *et al.*, 2002) não são satisfeitas.

Sincronização inexata

Conforme mencionado na Seção 4.2.3, caso o modelo que descreve o comportamento livre do sistema a ser controlado satisfaça a propriedade definida por MALIK (2002), então, é possível realizar implementação sem precisar considerar os aspectos da comunicação entre o sistema a ser controlado e o sistema de controle. Em caso contrário, é necessário verificar se o comportamento obtido sob supervisão satisfaz as propriedades definidas por BALEMI (1992a) ou por PARK e CHO (2006). Caso nenhuma destas propriedades seja verificada, então, o problema denominado sincronização inexata pode se manifestar.

Detalhamento da abstração

A modelagem e implementação dos procedimentos operacionais permite realizar o detalhamento da abstração adotada na representação do comportamento livre do sistema a ser controlado.

6. Modelo de comunicação entre CLPs

No Capítulo 4 foi realizada uma breve abordagem aos sistemas distribuídos, tendo sido apresentada a importância da comunicação entre controladores na distribuição do controle. Foram apresentados os principais aspectos da tecnologia *Messaging*, a qual permite a comunicação entre controladores através da troca de mensagens. Ainda no referido capítulo, foram apresentados os blocos de comunicação definidos pela norma internacional IEC 61131-5 (IEC, 2000) empregados na troca de mensagens entre CLPs. Foi, também, conceituada a comunicação confiável.

Este capítulo apresenta um modelo de comunicação entre CLPs. Tal modelo é fundamentado nos blocos de comunicação mencionados no parágrafo anterior. Além de garantir que a comunicação entre CLPs é confiável, este modelo garante, ainda, a satisfação de um conjunto de propriedades de interesse à distribuição do controle.

Na seção inicial do capítulo é apresentada uma conceituação geral do modelo de comunicação, na segunda seção são relacionadas as propriedades de interesse à distribuição do controle, bem como as ações a serem executadas caso alguma propriedade seja violada. Na terceira seção são detalhados os *Function Blocks* que estabelecem a comunicação e inter-relação entre CLPs. Finalmente, na última seção são apresentadas considerações finais sobre o modelo.

6.1 Conceituação geral do modelo

O modelo de comunicação definido neste capítulo prevê uma configuração elementar, a qual é constituída por dois CLPs entre os quais deve ser estabelecida a comunicação unidirecional. Na Figura 6.1 são ilustrados os aspectos principais desta configuração. O modelo considera que há disponível um sistema de comunicação que estabelece um canal de comunicação unidirecional interligando os CLPs entre os quais deve ser estabelecida a comunicação. Nesta configuração, tais CLPs são denominados Emissor e Receptor. A comunicação é realizada através do envio de mensagens do CLP Emissor para o CLP Receptor. Este modelo é fundamentado em dois *Function Blocks* de comunicação, denominados Sender e Receiver. No CLP emissor é implementada uma instância do FB Sender e no CLP receptor uma instância do FB Receiver. Na referida figura estas instâncias são denominadas, respectivamente, SendFromXToY e ReceiveFromXToY.

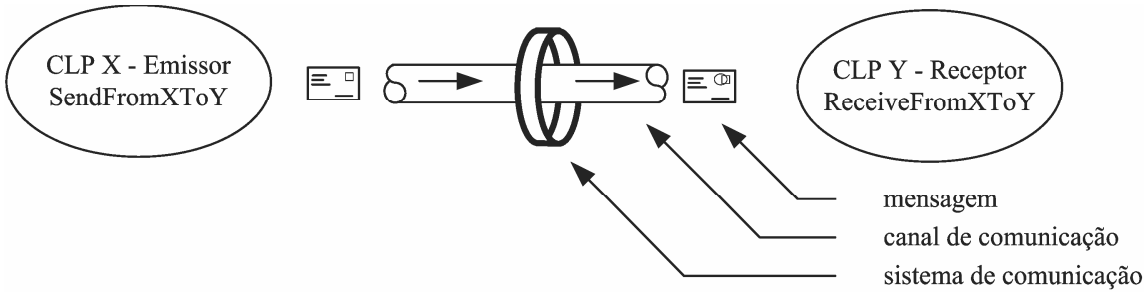


Figura 6.1 – Configuração elementar de comunicação

Na Seção 4.1.3 foram apresentadas as cinco etapas da tecnologia *Messaging* para comunicação através da troca de mensagens. Neste contexto, o programa de aplicação do CLP Emissor produz os dados que devem ser transmitidos para o CLP Receptor. O FB Sender é responsável por criar as mensagens e adicionar tais dados às mesmas. Este FB também é responsável por adicionar as mensagens ao canal de comunicação. O FB Receiver é responsável por ler as mensagens do canal de comunicação, extrair os dados relevantes e disponibilizá-los ao programa de aplicação do CLP Receptor.

Como etapa intermediária entre as descritas no parágrafo anterior, há a transferência da mensagem do CLP Emissor para o CLP Receptor. O sistema de comunicação deve estabelecer o canal de comunicação e executar esta tarefa com sucesso. Porém, o modelo de comunicação proposto abstrai os detalhes relativos a esta etapa.

A aplicação do modelo de comunicação não se restringe à configuração elementar ilustrada na Figura 6.1. Sua expansão para aplicação em configurações complexas é trivial. Caso a comunicação entre os CLPs deva ser bi-direcional, ou envolvendo um maior número de CLPs, então o sistema de comunicação deve estabelecer os canais de comunicação apropriados, além disto, devem ser implementadas adequadamente instâncias dos FBs Sender e Receiver nos diversos CLPs que constituem a rede de CLPs.

Sejam Z e W dois CLPs em uma rede qualquer. O sistema de comunicação deve estabelecer um canal de comunicação unidirecional interligando o CLP Z ao CLP W se e somente se é previsto o envio de mensagens do CLP Z para o CLP W. Além disto, no CLP Z deve ser implementada uma instância do FB Sender específica para o envio de mensagens para o CLP W, no CLP W deve ser implementada uma instância do FB Receiver específica para receber mensagens do CLP Z, se e somente se é previsto o envio de mensagens do CLP Z para o CLP W. Toda mensagem enviada do CLP Z para o CLP W deve utilizar o canal de comunicação e as instâncias dos FBs mencionadas acima. Diz-se que a instância do FB Sender implementada no CLP Z é parceira de comunicação da instância do FB Receiver implementada no CLP W, e vice-versa.

Exemplo 6.1)

Considere a rede constituída por quatro CLPs (CLP-0, CLP-1, CLP-2 e CLP-3) ilustrada na Figura 6.2. O CLP-0 deve ser capaz de enviar mensagens para todos os demais CLPs da rede. Este CLP também deve ser capaz de receber mensagens de todos os demais CLPs. Além disto, o CLP-1, o CLP-2 e o CLP-3 não trocam mensagens entre si. Cada um dos CLPs é, simultaneamente, um emissor e um receptor de mensagens. Em particular, o CLP-0 é o emissor e receptor para o CLP-1, o CLP-2 e o CLP-3. O CLP-1 é emissor e receptor apenas para o CLP-0.

O sistema de comunicação deve estabelecer um canal de comunicação unidirecional que leva mensagens do CLP-0 para cada um dos demais CLPs (canais 0/1, 0/2 e 0/3). Também deve estabelecer um canal de comunicação que leva mensagens de cada um dos demais CLPs na rede para o CLP-0 (canais 1/0, 2/0 e 3/0). Isto totaliza seis canais de comunicação. Não são necessários canais de comunicação entre o CLP-1, o CLP-2 e o CLP-3.

No CLP-0 devem ser implementadas três instâncias do FB Sender. Na Figura 6.2 estas instâncias são denominadas SendFrom0To1, SendFrom0To2 e SendFrom0To3. Também devem ser implementadas três instâncias do FB Receiver, denominadas ReceiveFrom1To0, ReceiveFrom2To0 e ReceiveFrom3To0.

No CLP-1 deve ser implementada uma instância do FB Sender e outra do FB Receiver, denominadas SendFrom1To0 e ReceiveFrom0To1.

O par de instâncias SendFrom0To1 e ReceiveFrom0To1 permite o envio de mensagens do CLP-0 para o CLP-1. A instância ReceiveFrom0To1 é o parceiro de comunicação da instância SendFrom0To1, e vice-versa. O par de instâncias SendFrom1To0 e ReceiveFrom1To0 permite o envio de mensagens do CLP-1 para o CLP-0.

Na Figura 6.2 são apresentadas as instâncias dos FBs Sender e Receiver implementadas no CLP-2 e no CLP-3.

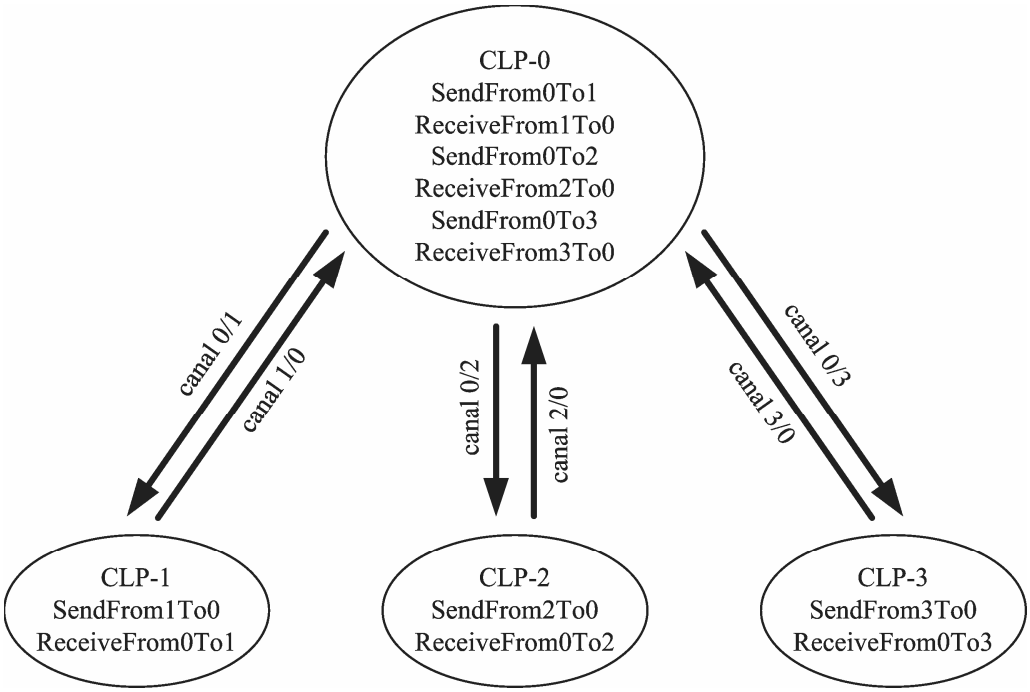


Figura 6.2 – Rede de CLPs

♦ fim do exemplo 6.1 ♦

6.2 Propriedades de interesse à distribuição do controle

Para garantir a correta comunicação e interação entre os CLPs é necessário que seja satisfeito um conjunto de propriedades de interesse. A seguir são relacionadas as propriedades consideradas relevantes para a distribuição do controle. Tais propriedades são sempre apresentadas considerando a comunicação unidirecional entre os pares parceiros de comunicação.

i) O início da comunicação ocorre quando, tanto o CLP emissor quanto o CLP receptor iniciam localmente o serviço. O início e/ou reinício da comunicação deve ser independente da ordem em que o CLP emissor e o CLP receptor iniciam ou reiniciam localmente o serviço;

ii) Iniciada a comunicação, deve ser preservada a vivacidade do emissor e do receptor;

iii) Mensagens devem ser enviadas sempre que houver novos dados e o sistema esteja apto para o envio;

iv) A integridade do conteúdo das mensagens deve ser preservada;

v) O ordenamento das mensagens deve ser preservado, não deve haver perda nem duplicação de mensagens;

vi) Dados incorporados às mensagens devem ser devidamente tratados, tanto no emissor quanto no receptor;

vii) Deve ser possível definir diferentes graus de prioridade para as mensagens.

Sempre que uma mensagem é recebida com sucesso, ou seja, o conjunto de propriedades acima é preservado, o FB Receiver deve extrair o conteúdo da mensagem e disponibilizar os dados ao programa de aplicação do CLP receptor.

Caso ocorra a violação de alguma das propriedades anteriores devem ser adotadas ações específicas. Os itens a seguir descrevem a possível causa da violação do conjunto de propriedades e as ações a serem adotadas:

i) Recepção duplicada de mensagem:

- descartar o conteúdo da mensagem duplicada;

ii) Perda de mensagem ou adulteração do conteúdo da mensagem:

- descartar o conteúdo da mensagem que identifica a perda de mensagens anteriores;

- descartar o conteúdo da mensagem adulterada;

- descartar as próximas mensagens de baixa prioridade até que ocorra o reinício do serviço de comunicação;

- o receptor deve sinalizar a ocorrência de erro;

iii) Erro no envio de mensagens ou violação da vivacidade do receptor:

- o emissor deve sinalizar a ocorrência de erro;

- suspender o envio de novas mensagens até o reinício do serviço de comunicação;

iv) Erro na recepção de mensagens ou violação da vivacidade do emissor:

- o receptor deve sinalizar a ocorrência de erro;

- descartar as próximas mensagens de baixa prioridade até que ocorra o reinício do serviço de comunicação;

- receber e tratar as próximas mensagens de alta prioridade.

6.3 Os *Function Blocks* de comunicação

Conforme apresentado na Seção 6.1, o envio e a recepção de mensagens é realizada através de instâncias dos FBs Sender e Receiver. Os SFCs apresentados nas Figuras 6.3 e 6.4 constituem a seção de código de tais FBs. As ações associadas a cada um dos passos destes SFCs são apresentadas nas Figuras 6.5 e 6.6.

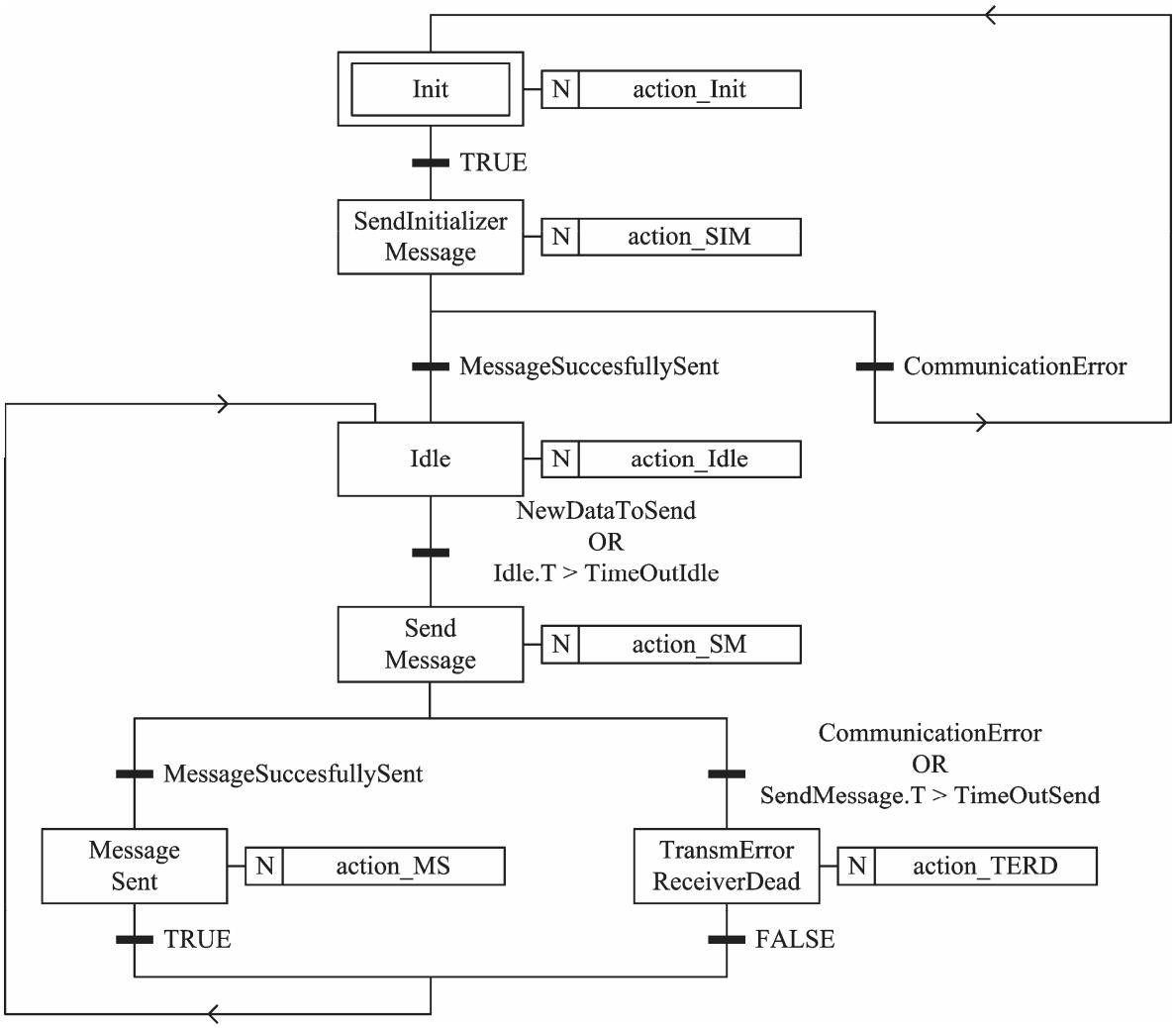


Figura 6.3 – SFC Sender

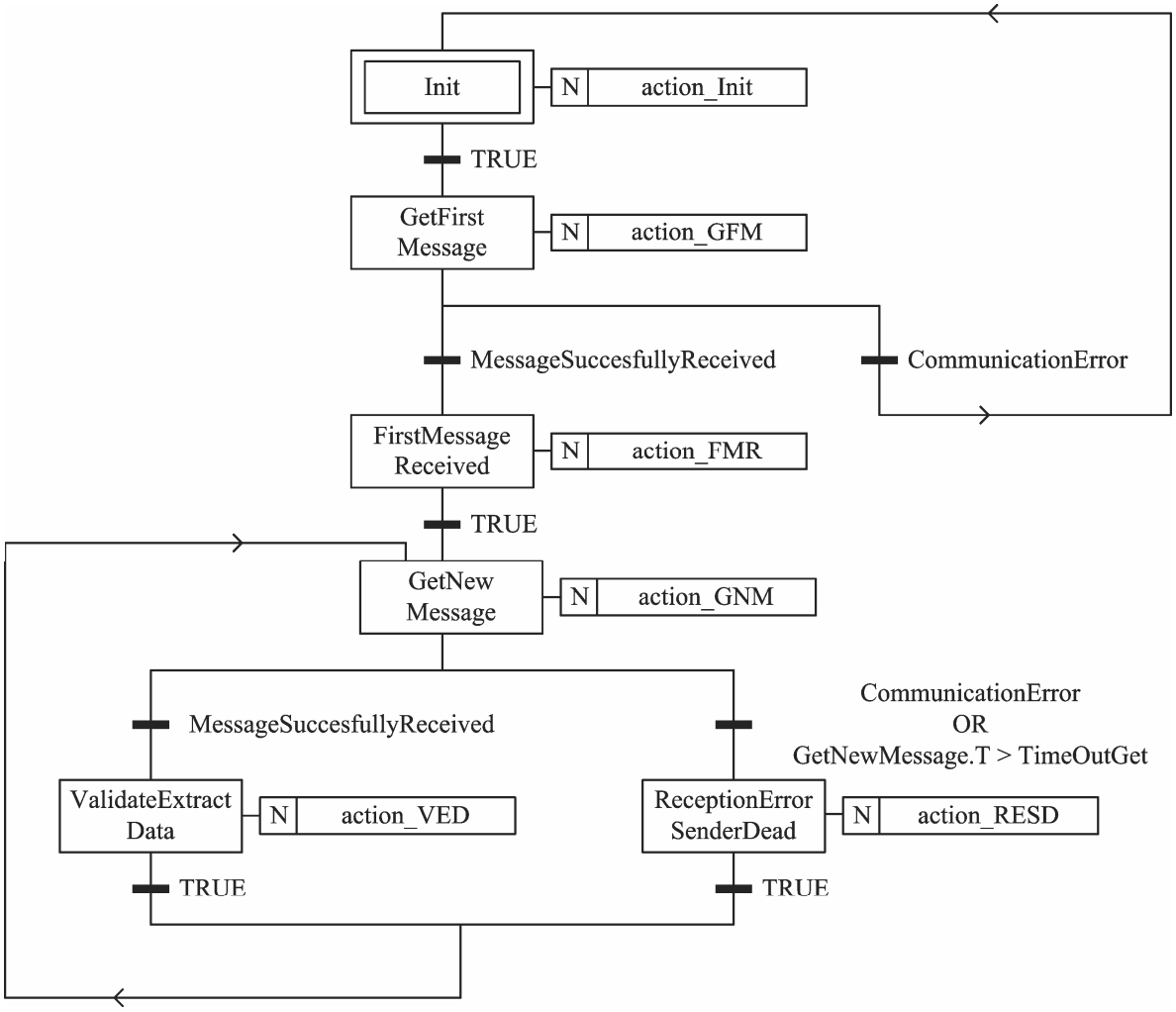


Figura 6.4 – SFC Receiver

action_Init

```
SendError := FALSE;
Status := 0;
ResetMessage;
```

action_SIM

```
IF SendIntializerMessage.T = 0 THEN
    MsgSeq := 0;
    Message.msg1 := MsgSeq;
    SendToY(REQ := TRUE,
            ID := ID,
            R_ID := RID,
            SD_1 := Message);
ELSE
    SendToY(REQ := FALSE,
            ID := ID,
            R_ID := RID,
            SD_1 := Message);
    MessageSuccesfullySent := SendToY.DONE;
    CommunicationError := SendToY.ERROR;
    Status := SendToY.Status;
END_IF;
```

action_Idle

```
IF Idle.T = 0 THEN
    MsgSeq := MsgSeq + 1;
END_IF;
```

action_SM

```
IF SendMessage.T = 0 THEN
    AddDataToMessage;
    SendToY(REQ := TRUE,
            ID := ID,
            R_ID := RID,
            SD_1 := Message);
ELSE
    SendToY(REQ := FALSE,
            ID := ID,
            R_ID := RID,
            SD_1 := Message);
    MessageSuccesfullySent := SendToY.DONE;
    CommunicationError := SendToY.ERROR;
    Status := SendToY.Status;
END_IF;
```

action_MS

```
IF MsgSeq >= MsgSeqMax THEN
    MsgSeq := 0;
END_IF;
```

action_TERD

```
SendError := TRUE;
```

Figura 6.5 – Ações associadas aos passos do SFC Sender

action_Init

```
RcvError := FALSE;
Status := 0;
ResetMessage;
```

action_GFM

```
RcvFromX(EN_R:= TRUE,
         ID := ID,
         R_ID := RID);
MessageSuccesfullyReceived := RcvFromX.NDR;
CommunicationError := RcvFromX.ERROR;
Message := RcvFromX.RD_1;
Status := RcvFromX.Status;
```

action_FMR

```
CheckIntegrity;
IF IntegrousMessage THEN
    (* Integrous Message *)
    ExtractDataFromMessage;
    MsgSeq := MsgSeq + 1;
ELSE
    (* Corrupted Message *)
     RcvError := TRUE;
    STATUS := 2;
END_IF;

IF MsgSeq > MsgSeqMax THEN
    MsgSeq := 1;
END_IF;
```

action_GNM

```
RcvFromX(EN_R:= TRUE,
         ID := ID,
         R_ID := RID);
MessageSuccesfullyReceived := RcvFromX.NDR;
CommunicationError := RcvFromX.ERROR;
Message := RcvFromX.RD_1;
Status := RcvFromX.Status;
```

action_VED

```
(* High Priority Messages *)
IF Message.msg1 = 0 THEN
    (* Initializer Message *)
    MsgSeq := 1;
    RcvError := FALSE;
    Status := 0;
    RETURN;
END_IF;

(* Low Priority Messages *)
IF NOT RcvError THEN
    (* No errors up to now *)
    IF Message.msg1 = MsgSeq THEN
        (* Message Order Correct *)
        CheckIntegrity;
        IF IntegrousMessage THEN
            (* Integrous Message *)
            ExtractDataFromMessage;
            MsgSeq := MsgSeq + 1;
        ELSE
            (* Corrupted Message *)
            RcvError := TRUE;
            STATUS := 2;
        END_IF;
    ELSIF Message.msg1 < MsgSeq THEN
        (* Duplicated Message *)
    ELSIF Message.msg1 > MsgSeq THEN
        (* Message Lost *)
        RcvError := TRUE;
        STATUS := 3;
    END_IF;
END_IF;

IF MsgSeq > MsgSeqMax THEN
    MsgSeq := 1;
END_IF;
```

action_RESD

```
RcvError := TRUE;
IF NOT CommunicationError THEN
    STATUS := 1;
END_IF;
```

Figura 6.6 – Ações associadas aos passos do SFC Receiver

O modelo de comunicação considera que o sistema de comunicação disponibiliza ao usuário o serviço *Programmed data acquisition*. Este serviço consiste basicamente dos pares de *Communication Function Blocks* USEND e URCV definidos na norma internacional IEC 61131-5

(IEC, 2000) e discutidos na Seção 4.1.4. Nas Figuras 6.5 e 6.6 SendToY é uma instância do FB USEND e RcvFromX é uma instância do FB URCV.

Caso o sistema de comunicação efetivamente utilizado não disponibilize o serviço *Programmed data acquisition* conforme definido na referida norma internacional, então as ações associadas aos FBs Sender e Receiver devem ser devidamente adaptadas em função do serviço de comunicação disponibilizado pelo referido sistema.

Uma mensagem pode conter apenas uma única informação (um único dado útil) ou um conjunto de informações (vários dados úteis). Conforme mencionado na Seção 4.1.3, uma mensagem é um pacote estruturado de dados, sendo que cada informação transmitida através da mensagem ocupa um campo desta estrutura.

Quando este modelo de comunicação é utilizado para realizar o envio de mensagens entre dois CLPs de uma rede, devem ser estabelecidas todas as possíveis informações (dados úteis) a serem anexadas às mensagens entre tais CLPs e previsto um campo da estrutura específico para cada dado. A cada mensagem enviada, as informações a serem anexadas determinam o valor a ser estabelecido a cada um destes campos. Toda mensagem enviada deve ter a mesma estrutura. A estrutura de uma mensagem pode ser considerada como um formulário, especialmente elaborado para estabelecer a comunicação entre os CLPs em questão. Toda mensagem entre estes CLPs emprega o mesmo formulário. O envio de uma mensagem corresponde ao envio de uma nova via do formulário devidamente preenchida.

Para poder avaliar o ordenamento do recebimento das mensagens, bem como a integridade do conteúdo das mesmas, cada mensagem deve conter dois campos adicionais aos campos necessários para transmitir os dados úteis. O primeiro campo é um número inteiro que representa o número seqüencial das mensagens. O segundo campo é um valor numérico empregado para avaliar a integridade da mensagem.

Nas Figuras 6.7 e 6.8 é apresentada a seção de declaração de variáveis dos FBs Sender e Receiver na forma elementar. Nestas figuras a definição da estrutura de dados MSG especifica a estrutura básica da mensagem, sem contudo incluir os campos correspondentes aos dados úteis a serem transmitidos.

As ações apresentadas nas Figuras 6.5 e 6.6, as seções de declaração de variáveis, bem como, a estrutura de dados da mensagem devem ser devidamente expandidas. Tal expansão é determinada, tanto pela especificação do conteúdo da mensagem, quanto pela forma como estes dados são trocados entre os demais POUs do programa de aplicação e os FBs Sender e Receiver.

```
FUNCTION_BLOCK Sender

(* Seção de declaração de variáveis *)

TYPE                                    (* Definição de tipos de variáveis *)
  MSG :                                 (* estrutura da mensagem *)
    STRUCT
      msg1 : INT;                       (* sequenciador de mensagens *)
      msg2 : INT;                       (* verificador de integridade *)
    END_STRUCT;
END_TYPE

VAR_INPUT
  ID: COMM_CHANNEL;                     (* ident. do canal de comunic.*)
  R_ID : STRING;                        (* ident. do parceiro de comunicação*)
END_VAR

VAR_OUT
  SendError: BOOL;                      (* erro no envio de mensagem *)
  Status: INT;                          (* especificação do erro *)
END_VAR

VAR
   SendToY: USEND;                      (* instância do FB USEND da IEC 61131-5
*)
  Message : MSG;                        (* mensagem *)
  MsgSeq : INT;                         (* sequenciador local de mensagens *)
  MsgSeqMax : INT;                      (* valor máx. do sequenciador de mensagens
*)
  TimeOutIdle, TimeOutSend : Time;
  MessageSuccessufullySent : BOOL;      (* mensagem enviada com sucesso *)
  CommunicationError : BOOL;            (* erro no envio da mensagem *)
  NewDataToSend : BOOL;                 (* novos dados a serem enviados *)
END_VAR

(* Fim da seção de declaração de variáveis *)
```

Figura 6.7 – Seção de declaração de variáveis do FB Sender

```
FUNCTION_BLOCK Receiver

(* Seção de declaração de variáveis *)

TYPE                                        (* Definição de tipos de variáveis *)
  MSG :                                     (* estrutura da mensagem *)
    STRUCT
      msg1 : INT;                           (* sequenciador de mensagens *)
      msg2 : INT;                           (* verificador de integridade *)
    END_STRUCT;
END_TYPE

VAR_INPUT
  ID: COMM_CHANNEL;                         (* ident. do canal de comunic.*)
  R_ID : STRING;                            (* ident. do parceiro de comunicação*)
END_VAR

VAR_OUT
  RcvError: BOOL;                           (* erro na recepção de mensagens *)
  Status: INT;                              (* especificação do erro *)
END_VAR

VAR
   RcvFromX : URCV;                         (* instância do FB URCV da IEC 61131-5 *)
  Message : MSG;                            (* mensagem *)
  MsgSeq : INT;                             (* sequenciador local de mensagens *)
  MsgSeqMax : INT;                          (* valor máx. do sequenciador de mensagens
*)
  TimeOutGet : Time;
  MessageSuccessufullyReceived : BOOL;      (* mensagem recebida com sucesso *)
  CommunicationError : BOOL;                (* erro na recepção da mensagem *)
  IntegrousMessage : BOOL;                  (* mensagem íntegra *)
END_VAR

(* Fim da seção de declaração de variáveis *)
```

Figura 6.8 – Seção de declaração de variáveis do FB Receiver

No FB Sender há um seqüenciador local de mensagens que armazena o número da próxima mensagem a ser enviada. No FB Receiver o seqüenciador local de mensagens armazena o número da próxima mensagem a ser recebida. Nas Figuras 6.5 a 6.8 a variável MsgSeq implementa o seqüenciador local de mensagens.

Quando a mensagem é criada (action_SIM ou action_SM), o FB Sender adiciona o valor do seqüenciador local de mensagens ao primeiro campo da mensagem. Este seqüenciador deve ser devidamente incrementado cada vez que uma mensagem é enviada. Quando o FB Receiver extrai os dados da mensagem (action_VED), é realizada a comparação do valor armazenado no primeiro campo da mensagem com o seqüenciador local de mensagens deste FB. Com base no resultado desta comparação é possível determinar se o ordenamento das mensagens está preservado, se houve

perda ou se houve duplicação de mensagens. O FB Receiver deve incrementar devidamente o seu seqüenciador local cada vez que uma mensagem é recebida.

O valor numérico empregado para avaliar a integridade da mensagem é produzido por um algoritmo matemático. Este algoritmo deve ser implementado tanto no FB Sender, quanto no FB Receiver. Quando a mensagem é criada, este algoritmo deve ser alimentado com os dados úteis a serem transmitidos. O valor de retorno do algoritmo deve ser adicionado ao segundo campo da mensagem. Quando a mensagem é lida, os dados úteis devem ser extraídos da mensagem e utilizados para alimentar o mesmo algoritmo. O valor de retorno obtido nesta execução do algoritmo deve ser comparado com o segundo campo da mensagem. Em caso de igualdade é constatada a integridade dos dados.

Caso o sistema de comunicação efetivamente utilizado incorpore a função de verificação da integridade das mensagens, então os FBs Sender e Receiver não precisam realizar esta operação. Neste caso, toda mensagem recebida pelo FB Receiver é considerada íntegra.

Por considerar-se que os sistemas de comunicação adotados nos CLPs atuais usualmente incorporam a função de verificação da integridade das mensagens, não foram dedicados maiores esforços na busca ou desenvolvimento de um algoritmo que realize a verificação de mensagens. Em (STALLINGS, 1999) são apresentados algoritmos desenvolvidos para este fim e que podem ser adaptados para a estrutura da mensagem a ser transmitida.

O FB Sender inicia localmente a comunicação através da ativação do passo Init. Quando ocorre a ativação do passo SendInitializerMessage é criada e enviada uma mensagem inicializadora. O único dado relevante desta mensagem é o seqüenciador de mensagens com o valor zero.

De forma similar, o FB Receiver inicia localmente a comunicação através da ativação do passo Init. O recebimento de mensagens é habilitado quando ocorre a ativação do passo GetFirstMessage.

Iniciada a comunicação, sempre que o passo Init do SFC Sender é novamente ativado, é enviada uma mensagem inicializadora. Sempre que o FB Receiver identifica o recebimento de uma mensagem inicializadora (na ação action_VED Message.msg1 = 0 resulta em VERDADEIRO) é estabelecido o valor inicial às variáveis que sinalizam erro na comunicação (RcvError := FALSE, STATUS := 0) e à variável associada ao seqüenciador local de mensagens do FB Receiver (MsgSeq := 1).

Sempre que o passo Init do SFC Receiver é novamente ativado, ao seqüenciador local de mensagens deste FB é estabelecido o valor correspondente constante na primeira mensagem recebida após a ativação do passo GetFirstMessage.

Qualquer das ações mencionadas nos dois parágrafos anteriores resulta no reinicio da comunicação. O início e/ou reinício da comunicação resulta na exclusão da sinalização de ocorrência erro nos FBs Sender e Receiver (SendError := FALSE e RcvError := FALSE), bem como na sincronização dos seqüenciadores locais de mensagens destes FBs.

Iniciada a comunicação, os FBs Sender e Receiver verificam continuamente a vivacidade do parceiro de comunicação.

Se o passo ativo do SFC Sender é o passo Idle e não há novos dados a serem transmitidos, então uma mensagem com valores nulos associados aos campos correspondentes aos dados úteis é automaticamente criada e enviada assim que transcorrer um tempo pré-estabelecido (TimeOutIdle). Isto sinaliza a vivacidade deste elemento na rede. Caso o passo ativo do SFC Receiver seja o passo GetNewMessage e não seja recebida qualquer mensagem ao longo de um tempo pré-estabelecido (TimeOutGet) é identificada a falta de vivacidade do parceiro de comunicação. Quando o FB Receiver extrai os dados da mensagem com valores nulos e os transfere ao programa de aplicação do CLP receptor não há alterações no estado deste programa.

Sempre que uma mensagem é enviada (o passo ativo do SFC Sender é o passo SendMessage) é esperado que o sistema de comunicação sinalize que a mensagem foi enviada com sucesso (MessageSuccesfullySent). Caso esta sinalização não ocorra ao longo de um tempo pré-estabelecido (TimeOutSend) é identificada a falta de vivacidade do parceiro de comunicação.

Caso ocorra erro no envio de uma mensagem (o passo ativo do SFC Sender é o passo SendMessage e o sistema de comunicação sinaliza CommunicationError) ou seja identificada a falta de vivacidade do parceiro de comunicação (SendMessage.T > TimeOutSend resulta VERDADEIRO), então o FB Sender interrompe o envio de mensagens até que ocorra o reinício do serviço (o passo ativo do SFC Sender passa a ser o passo TransmErrorReceiverDead).

Quando o passo ativo do SFC Sender é o passo TransmErrorReceiverDead o envio de mensagens é interrompido. Como conseqüência o FB Receiver não irá identificar a vivacidade do parceiro de comunicação, o que resulta na ocorrência de erro neste FB.

Para o estabelecimento dos valores das variáveis TimeOutIdle, TimeOutSend e TimeOutGet dever haver um comprometimento entre a rapidez com que é desejável detectar a falta de vivacidade do parceiro de comunicação, os parâmetros que determinam o desempenho da rede, a quantidade de dados que são transmitidos em cada mensagem e o número de elementos ativos na rede.

O valor atribuído à variável TimeOutGet especifica o menor valor de tempo necessário para identificar a falta de vivacidade do emissor. O valor atribuído à variável TimeOutSend especifica o menor valor de tempo necessário para identificar a falta de vivacidade do receptor.

Uma análise informal e simplificada do caso em que há apenas um emissor e um receptor na rede permite concluir que:

i) TimeOutGet deve ser maior do que (TimeOutIdle + T1 + T2 + 2*T3);

ii) TimeOutSend deve ser maior do que (T1 + T2 + T3);

onde:

– T1: tempo necessário para o sistema de comunicação enviar a mensagem;

– T2: tempo necessário para o sistema de comunicação notificar que a mensagem foi enviada com sucesso;

– T3: tempo do ciclo de atualização do CLP emissor.

É importante notar que o FB Sender só está apto a enviar mensagens quando o passo ativo neste FB é o passo Idle. Se esta situação não é verificada e novos dados são produzidos pelo programa de aplicação do CLP onde está implementado o FB Sender, então uma mensagem contendo estes dados não poderá ser enviada imediatamente. Tais dados devem ser preservados pelo programa de aplicação ou memorizados pelo FB Sender até que uma nova mensagem possa ser enviada.

O grau de prioridade das mensagens é estabelecido pelo valor associado a cada um dos campos da mensagem. Na ação action_VED associada ao passo ValidateExtractData do SFC Receiver, o conteúdo das mensagens de baixa prioridade é tratado somente se não foi identificada a ocorrência de erros anteriores ("NOT RcvError" resulta em VERDADEIRO). O conteúdo das mensagens de alta prioridade é sempre tratado. Esta ação é, então, subdivida em duas seções. Na primeira seção são especificadas as ações correspondentes às possíveis mensagens de alta prioridade. Na segunda seção são especificadas as ações correspondentes às ações de baixa prioridade. Na forma como está implementada a ação action_VED (Figura 6.6) há apenas uma mensagem de alta prioridade, a qual é a mensagem inicializadora. Sempre que o primeiro campo da mensagem carrega o valor zero ("Message.msg1 = 0" resulta em VERDADEIRO) é identificado que esta é uma mensagem inicializadora da comunicação. Toda mensagem em que o primeiro campo carrega um valor diferente de zero é considera uma mensagem de baixa prioridade. O modelo de comunicação pode ser expandido definindo outras mensagens de alta prioridade. Para tanto, devem ser adicionadas na primeira seção da ação action_VED ações executadas condicionalmente ao valor de determinados campos da mensagem.

Dentre as propriedades de interesse relacionadas na seção anterior não há qualquer requisito de desempenho. É possível que a adoção de um determinado sistema de comunicação resulte em índices de erros na comunicação insatisfatórios. Tais erros podem ser ocasionados tanto

pela violação da integridade quanto pela violação do ordenamento das mensagens. Caso sejam estabelecidos critérios de desenpenho, então este modelo pode ser expandido de forma a permitir a retransmissão de mensagens. Também podem ser implementados mecanismos para reordenamento de mensagens. Tais ações mascaram a ocorrência de falhas, convertendo falhas arbitrárias em falhas por omissão.

Nas ações apresentadas nas Figuras 6.5 e 6.6 os itens ResetMessage, AddDataToMessage, ExtractDataFromMessage e CheckIntegrity encapsulam ações que dependem da estrutura da mensagem a ser transmitida. O item ResetMessage deve estabelecer o valor nulo a todos os campos da mensagem; o item AddDataToMessage estabelece o valor a cada campo da mensagem em função do conteúdo a ser transmitido; o item ExtractDataFromMessage lê o valor de cada campo da mensagem e os disponibiliza ao programa da aplicação do CLP receptor; finalmente, o item CheckIntegrity verifica a integridade da mensagem conforme mencionado no início desta seção.

6.4 Conclusão

Conforme discutido no Capítulo 4, a comunicação entre controladores é um aspecto fundamental na distribuição do controle, contudo, ao longo da pesquisa bibliográfica realizada, não foi encontrada qualquer referência que aborde modelos de comunicação entre CLPs. O modelo de comunicação proposto neste capítulo foi incorporado ao método de implementação distribuída da arquitetura de controle supervisório proposta por QUEIROZ e CURY (2002), o qual é apresentado no próximo capítulo.

Este modelo de comunicação foi avaliado com sucesso nas configurações apresentadas nas Figuras 6.1 e 6.2 com CLPs da marca SIEMENS. Para tanto os CLPs foram interligados através de uma rede de protocolo proprietário denominado MPI. Visto que o sistema de comunicação utilizado não disponibiliza o serviço *Programmed data acquisition*, foi necessário realizar pequenas adaptações nos FBs de comunicação para utilização do serviço de comunicação efetivamente disponibilizado em cada caso.

7. Método para implementação distribuída do controle supervisório

No Capítulo 1 discutiu-se a importância da distribuição do controle, tendo sido ressaltados aspectos como desempenho, distribuição espacial, facilidade de integração e reutilização. No Capítulo 4 foram abordados os principais desafios encontrados na implementação distribuída, bem como os possíveis tratamentos para enfrentar tais problemas. Do apresentado no Capítulo 4 verifica-se a intensa atividade do setor acadêmico na investigação da aplicabilidade da norma internacional IEC 61499. Porém, a incompatibilidade desta norma com as características operacionais dos CLPs atuais é um fator que limita sua aplicabilidade imediata. No Capítulo 5 foi proposto um método que permite a implementação, concentrada em um único CLP, da arquitetura de controle supervisório proposta por QUEIROZ e CURY (2002). No Capítulo 6 foi proposto um modelo de comunicação entre CLPs, necessário para a distribuição do controle.

Este capítulo estende o método apresentado no Capítulo 5 de forma a permitir que a implementação do controle seja distribuída em um conjunto de CLPs.

Na seção inicial do capítulo é introduzido o conceito da distribuição da implementação do controle. Apresenta-se o objetivo de realizar a distribuição do controle e no que ela consiste. Apresentam-se, também, as restrições impostas à realização da distribuição do controle. Na segunda seção apresenta-se uma visão geral do método. Na terceira seção são detalhados os aspectos de comunicação e inter-relação entre os CLPs. Na quarta seção são detalhados os *Function Blocks* que garantem que os aspectos detalhados na seção anterior são efetivamente alcançados. Finalmente, na última seção apresentam-se considerações sobre o método.

7.1 Conceituação da distribuição

Na Figura 4.13 foi apresentada a Arquitetura de Controle proposta por Queiroz e Cury. O método de implementação apresentado no Capítulo 5 considera que o sistema de controle é constituído por um único CLP. Neste CLP são integralmente implementados os níveis Supervisores Modulares, Sistema Produto e Procedimentos Operacionais.

A distribuição da implementação da arquitetura referida no parágrafo anterior visa atingir os seguintes objetivos:

i) melhorar o desempenho do sistema;

ii) realizar a distribuição espacial do controle;

iii) realizar a integração de controladores que apresentem características específicas de interface com o sistema a ser controlado;

iv) viabilizar a utilização dos controladores disponíveis para a implementação do controle, considerando a limitação de recursos de cada um dos mesmos;

v) possibilitar a reutilização de uma estratégia de controle já implementada e validada em um determinado controlador;

vi) simplificar o processo de reconfiguração do sistema;

vii) preservar a modularidade natural do sistema a ser controlado.

A distribuição do controle pode ser realizada implementando em diferentes CLPs os elementos que constituem os níveis Supervisores Modulares, Sistema Produto e Procedimentos Operacionais da arquitetura de controle proposta por Queiroz e Cury. Com isto, é necessário estabelecer a comunicação entre os CLPs, bem como a coordenação do processamento do código implementado em cada CLP com relação ao processamento do código implementado nos demais CLPs da rede.

Sob o foco tratado neste capítulo, o sistema de controle é constituído por diversos CLPs, sendo um CLP coordenador (possivelmente designado mestre) e diversos CLPs executores (possivelmente designados escravos). A Figura 7.1 ilustra a arquitetura de controle sob esta perspectiva de distribuição. Em cada CLP executor é implementada uma parcela do nível Procedimentos Operacionais. No CLP coordenador são implementados integralmente o nível Supervisores Modulares e o nível Sistema Produto e, possivelmente, uma parcela do nível Procedimentos Operacionais.

O método de implementação concentrado, conforme apresentado no Capítulo 5, corresponde ao caso em que não há qualquer CLP executor e o nível procedimentos operacionais está integralmente implementado no CLP coordenador.

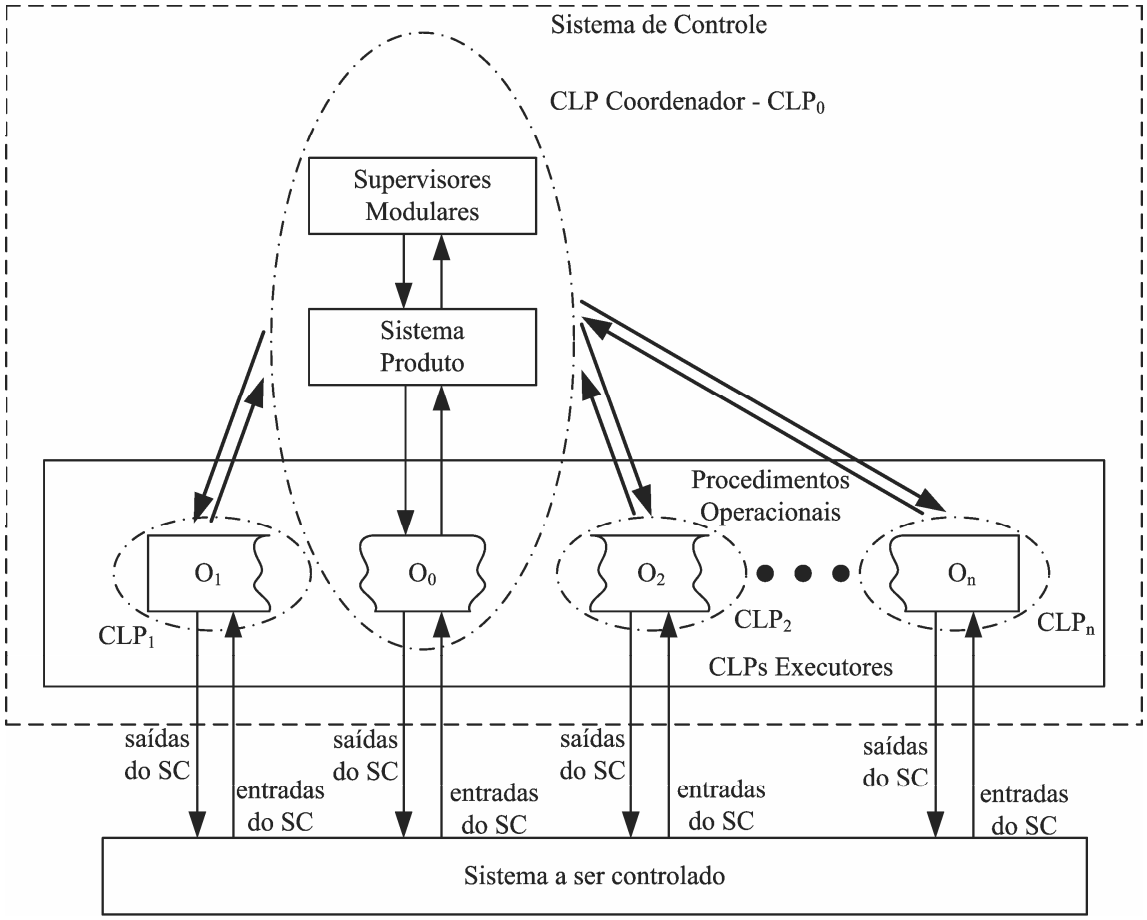


Figura 7.1 – Arquitetura de controle distribuída em múltiplos CLPs

Conforme apresentado na Seção 5.7, é possível que o conjunto de procedimentos operacionais seja expandido com a inclusão de procedimentos de inicialização. Designa-se o conjunto de procedimentos operacionais resultante como "O". Designa-se, então, $O_k \subseteq O$, com $k \in \{0, 1, 2, ..., n\}$, o subconjunto de procedimentos operacionais implementados no CLP_k.

O método de implementação distribuída impõe as seguintes restrições à fragmentação do nível Procedimentos Operacionais:

i) $\cup_{k=0}^{n} O_k = O$;

ii) $(\forall\ x, y \in \{0, 1, 2, ..., n\})\ O_x \cap O_y = \varnothing$.

As restrições acima especificam que cada procedimento operacional deverá estar implementado em um, e apenas um, CLP que constitui o sistema de controle. Destaca-se que é admissível o caso em que $O_0 = \varnothing$, tal que CLP_0 é o CLP coordenador.

Exemplo 7.1)

Conforme apresentado no Exemplo 5.11, o conjunto de procedimentos operacionais, expandido com os procedimentos de inicialização, associado ao sistema de manufatura apresentado no exemplo motivador é dado por O = {oa0, oa1, ot1, oa2, oa3, ot3, oa4, oa5} ∪ {iniG1A, iniG1B, iniG2, iniG3, iniG5A, iniG5B}.

Foi realizada com sucesso a implementação do controle do referido sistema de manufatura distribuída em 4 CLPs. Como CLP coordenador (CLP_0) foi utilizado um CLP SIEMENS 317-2 DP. Os CLPs CLP_1 e CLP_2 são dois CLPS SIEMENS 314-IFM e o CLP_3 é um CLP SIEMENS 315-2 DP. Para este conjunto de CLPs foi realizada a seguinte fragmentação do conjunto O: O_0 = {oa0}, O_1 = {oa1, ot1, iniG1A, iniG1B, oa3, ot3, iniG3, oa4}, O_2 = {oa2, iniG2}, O_3 = {oa5, iniG5A, iniG5B}.

♦ fim do exemplo 7.1 ♦

7.2 Visão geral do método

Conforme ilustrado através da Figura 7.1, o método de implementação distribuída prevê um CLP Coordenador (CLP_0) e diversos CLPs Executores (CLP_1, CLP_2, ..., CLP_n). Na rede de CLPs resultante da aplicação deste método deve ser estabelecida a comunicação entre o CLP coordenador e cada CLP executor, porém, não é necessária a comunicação entre os CLPs executores. A comunicação entre CLPs é realizada através da troca de mensagens.

Conforme apresentado nos Capítulos 4 e 5, sempre que é gerado um evento controlável no nível Sistema Produto, ocorre a ativação do comando correspondente. Este comando é ativado para disparar a execução das atividades pelo procedimento operacional associado ao evento em questão. Sempre que um procedimento operacional detecta a ocorrência de um evento não-controlável, a resposta associada ao evento em questão é devidamente incrementada. Quando o valor associado a uma resposta é maior do que zero, é realizado o tratamento do evento correspondente. Uma mensagem enviada do CLP coordenador para um CLP executor notifica este último sobre a ativação de comandos. Uma mensagem em sentido contrário notifica o número de ocorrências de eventos não-controláveis pendentes para tratamento. Também notifica sobre o processamento de comandos pelos procedimentos operacionais implementados no CLP executor. A estrutura detalhada destas mensagens é apresentada na Seção 7.3.

O envio de mensagens do CLP Coordenador para um CLP Executor é realizado através de especializações dos FBs Sender e Receiver introduzidos no capítulo anterior. Tais especializações são, respectivamente, denominadas SendCmd e RcvCmd. De forma similar, o envio de mensagens

de um CLP Executor para o CLP Coordenador é realizado através de outras duas especializações destes FBs. O FB SendRspCmdNtf é uma especialização do FB Sender e o FB RcvRspCmdNtf é uma especialização do FB Receiver. Estes FBs determinam, também, a inter-relação entre os CLPs.

Na Tabela 5.1 foi apresentada a estrutura do programa de aplicação resultante da adoção do método de implementação concentrada. A estrutura do programa de aplicação do CLP Coordenador é praticamente inalterada em relação à estrutura descrita na referida tabela. A principal alteração diz respeito ao conjunto de FBs que implementam os procedimentos operacionais. Ao invés de implementar o conjunto de procedimentos operacionais na íntegra, apenas o subconjunto O_0 deve ser implementado no CLP Coordenador. Desta forma, o FB OP deve realizar a chamada apenas aos FBs neste subconjunto. Em geral, neste CLP deve ser implementada uma instância do FB SendCmd e uma instância do FB RcvRspCmdNtf associada a cada CLP Executor.

Exemplo 7.2)

Considerando a distribuição conforme realizada no Exemplo 7.1, a Tabela 7.1 apresenta as instâncias de FBs implementadas em cada CLP para realizar a comunicação.

Tabela 7.1 – Instâncias dos FBs de comunicação

mensagem	CLP emissor	instância do FB SendCmd	CLP receptor	instância do FB RcvCmd
$CLP_0 \rightarrow CLP_1$	CLP_0	SendCmdTo1	CLP_1	RcvCmdFrom0
$CLP_0 \rightarrow CLP_2$	CLP_0	SendCmdTo2	CLP_2	RcvCmdFrom0
$CLP_0 \rightarrow CLP_3$	CLP_0	SendCmdTo3	CLP_3	RcvCmdFrom0
mensagem	CLP emissor	instância do FB SendRspCmdNtf	CLP receptor	instância do FB RcvRspCmdNtf
$CLP_1 \rightarrow CLP_0$	CLP_1	SendRspCmdNtfTo0	CLP_0	RcvRspCmdNtfFrom1
$CLP_2 \rightarrow CLP_0$	CLP_2	SendRspCmdNtfTo0	CLP_0	RcvRspCmdNtfFrom2
$CLP_3 \rightarrow CLP_0$	CLP_3	SendRspCmdNtfTo0	CLP_0	RcvRspCmdNtfFrom3

♦ fim do exemplo 7.2 ♦

No *Program* Main implementado no CLP coordenador são realizadas pequenas alterações em relação à versão descrita no Capítulo 5. Em todas as ações associadas aos passos do SFC Main (Figura 5.1) deve ser realizada a chamada a todas as instâncias dos FBs SendCmd e RcvRspCmdNtf implementadas no CLP coordenador. Na ação action_SI deve ser inicializado o SFC que constitui a seção de código de cada um destes FBs. Ao passo PSIted deve ser associada uma ação que, a cada ciclo de atualização do CLP, realiza a chamada a tais instâncias. A inclusão

desta ação é necessária para preservar a vivacidade dos elementos na rede enquanto este é o passo ativo do SFC Main.

A Figura 7.2 apresenta o SFC Exec, o qual constitui a seção de código do *Program* de mesmo nome. O passo ativo deste SFC estabelece o modo de operação do CLP executor no qual ele está implementado. Há quatro modos de operação distintos: *Software Initialization* (passo SI); *Run* (passo Run); *Emergency* (passo Emg) e *Idle* (passo init).

A Tabela 7.2 detalha a estrutura do programa de aplicação de cada CLP executor.

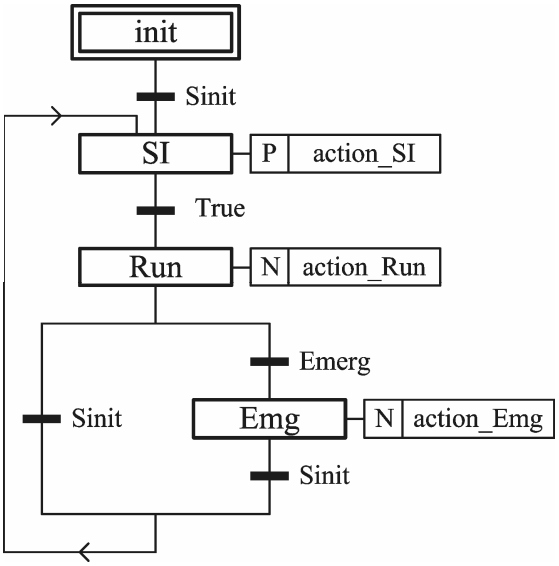


Figura 7.2 – SFC Exec

Tabela 7.2 – *Program Organization Units* do programa de aplicação do CLP executor - CLP_k, $k \in \{1, 2, ..., n\}$		
nome	tipo	comentários
$o \in O_k$	FB	Representa um procedimento operacional implementado no CLP_k
RcvCmd	FB	FB para o recebimento de mensagens oriundas do CLP coordenador
SendRspCmdNtf	FB	FB para o envio de mensagens para o CLP coordenador
Exec	Program	Estabelece o modo de operação do CLP executor e as ações correspondentes a cada um dos modos

Exemplo 7.3)

Conforme o Exemplo 7.1, O_1 = {oa1, ot1, iniG1A, iniG1B, oa3, ot3, iniG3, oa4}. Estes são os procedimentos operacionais implementados no CLP_1. Neste CLP devem ser implementadas a instância RcvCmdFrom0 do FB RcvCmd (para recebimento de mensagens do CLP coordenador) e a instância SendRspCmdNtfTo0 do FB SendRspCmdNtf (para envio de mensagens para o CLP coordenador).

♦ fim do exemplo 7.3 ♦

Quando o passo SI do SFC Exec é ativado, o CLP executor entra no modo *Software Initialization* e a ação associada a este passo (action_SI) é executada apenas uma vez. Esta ação deve inicializar o SFC que constitui a seção de código de cada um dos procedimentos operacionais em O_k. Ela também deve inicializar os SFCs nos FBs RcvCmd e SendRspCmdNtf. A inicialização destes SFCs permite o início da comunicação com o CLP coordenador. Esta ação também deve estabelecer a todas as variáveis do CLP o valor inicial apropriado. Tais valores são os apresentados no Capítulo 5.

O CLP executor evolui automaticamente para o modo *Run* após a execução da ação action_SI. Isto é realizado através da ativação do passo Run. Enquanto este passo está ativo a ação action_Run é executada uma vez a cada ciclo de atualização do CLP. Esta ação deve realizar a chamada seqüencial a todos os procedimentos operacionais em O_k. Esta ação também deve executar a chamada às instâncias dos FBs RcvCmd e SendRspCmdNtf para a devida comunicação com o CLP coordenador.

No modo *Emergency* a ação action_Emg deve ativar adequadamente os atuadores e alarmes do sistema. Esta ação também deve executar a chamada às instâncias dos FBs RcvCmd e SendRspCmdNtf para a devida comunicação com o CLP coordenador. No modo *Idle* nenhuma ação é executada.

No CLP coordenador, o passo ativo do SFC Main estabelece o modo de operação do sistema como um todo. Quando o passo ativo deste SFC é o passo PSI ou Man ou Sup, é possível que ocorra a ativação de comandos, determinando o disparo da execução de procedimentos operacionais. O CLP coordenador executa os procedimentos em O_0 e envia mensagens para os CLPs executores, notificando sobre a ativação de comandos, para que os procedimentos implementados nestes CLPs sejam executados. Tais procedimentos serão executados se o modo de operação do CLP executor for o modo *Run*. O número de ocorrências de eventos não-controláveis pendentes para tratamento que são detectadas pelos CLPs executores são notificadas ao CLP coordenador através do envio de mensagens.

Considere o CLP coordenador e um determinado CLP executor na rede. Caso o FB RcvCmd implementado no CLP executor sinalize a ocorrência de erro na recepção de mensagens, este erro será notificado ao CLP coordenador através do FB SendRspCmdNtf, também implementado no CLP executor. Caso ocorra um erro no envio de mensagens no FB SendRspCmdNtf, o SFC deste FB é conduzido para o passo TransmErrorReceiverDead e o envio de mensagens para o CLP coordenador é interrompido. O FB RcvRspCmdNtf implementado no CLP coordenador não irá identificar a vivacidade do seu parceiro de comunicação e irá notificar a ocorrência de erro. Desta forma, o CLP coordenador sempre identifica a ocorrência de erro nos FBs RcvCmd e SendRspCmdNtf implementados nos CLPs executores.

Quando ocorre um erro em qualquer instância dos FBs SendCmd ou RcvRspCmdNtf implementadas no CLP coordenador, ou quando este CLP identifica a ocorrência de erro nos FBs RcvCmd e SendRspCmdNtf implementados em qualquer CLP executor, o sistema é conduzido automaticamente para o modo manual (a variável Manual do SFC Main deve assumir valor VERDADEIRO). Com isto a geração de eventos controláveis é interrompida, porém a execução dos procedimentos operacionais, tanto nos CLPs executores quanto no CLP coordenador, não é interrompida. Caso ocorra um evento não-controlável e a comunicação do CLP executor para o CLP coordenador esteja ativa, a ocorrência deste evento será sinalizada ao CLP coordenador para o devido tratamento. Caso esta comunicação não esteja ativa, a ocorrência de eventos não-controláveis continuará sendo identificada e registrada nos CLPs executores para posterior notificação ao CLP coordenador.

Com o sistema no modo Manual, o operador pode diagnosticar a causa do erro e, se possível, repará-lo. Caso isto seja possível, a comunicação entre os CLPs pode ser reiniciada e o sistema reconduzido para o modo *Supervised*. As ocorrências de eventos não-controláveis que estejam registradas nos CLPs executores serão devidamente notificadas ao CLP coordenador para tratamento. Caso não seja possível reparar o erro, o sistema como um todo deve ser reiniciado.

7.3 A comunicação e a inter-relação entre CLPs

Considerando a Figura 7.1, o CLP_k, $k \neq 0$, é um CLP executor e o conjunto de procedimentos operacionais implementado neste CLP é designado O_k. Este conjunto inclui procedimentos específicos para conduzir o sistema para o estado inicial, os procedimentos de inicialização. Associados a estes procedimentos há um conjunto de comandos, designado C_k, o qual inclui tanto os comandos associados a eventos controláveis quanto os comandos de inicialização. O conjunto de respostas associadas a eventos não-controláveis cuja ocorrência é identificada por

algum procedimento em O_k está contido em R_k. Ao conjunto R_k são incluídas as respostas que sinalizam a conclusão de procedimentos de inicialização.

Exemplo 7.4)

Considerando a fragmentação do conjunto de procedimentos operacionais apresentada no Exemplo 7.1 e a relação de comandos e respostas apresentadas na Tabela 5.5, tem-se que o conjunto de comandos e respostas associados a cada CLP executor é:

C_1 = {cmda1, cmdt1, iniG1Astart, iniG1Bstart, cmda3, cmdt3, iniG3start, cmda4};

C_2 = {cmda2, iniG2start};

C_3 = {cmda5, iniG5Astart, iniG5Bstart};

R_1 = {rspb1, rspl1, rspm1, iniG1Aend, iniG1Bend, rspb3, iniG3end, rspb4, rspm4};

R_2 = {rspb2, rspl2, rspm2, iniG2end};

R_3 = {rspb5, rspl5, rspm5, iniG5Aend, iniG5Bend}.

♦ fim do exemplo 7.4 ♦

Uma mensagem do CLP coordenador para o CLP executor CLP_k informa quais comandos em C_k foram ativados no CLP coordenador desde o envio da última mensagem. Informa, também, o modo de operação deste CLP.

Uma mensagem do CLP_k para o CLP coordenador informa quantas ocorrências de cada evento não-controlável foram detectadas pelos procedimentos operacionais em O_k desde o envio da última mensagem, ou seja, o valor armazenado nas variáveis em R_k. Esta mensagem notifica ao CLP coordenador quais os comandos em C_k que foram devidamente processados pelos procedimentos operacionais correspondentes desde o envio da última mensagem. Informa, também, o modo de operação do CLP_k e se a instância do FB RcvCmd implementada no CLP_k sinaliza a ocorrência de erro.

Supondo que C_k = {cmd1, cmd2, ..., cmdn}, com $|C_k| = n$, a estrutura detalhada da mensagem do CLP coordenador para o CLP_k é como o apresentado a seguir:

- campo 1: seqüenciador de mensagens;

- campo 2: verificador de integridade ;

- campo 3: modo de operação do CLP coordenador;

- campo 4: sinalização da ativação do comando cmd1;

- campo 5: sinalização da ativação do comando cmd2;

...

- campo w: sinalização da ativação do comando cmdn (w = 3+n).

Supondo que C_k = {cmd1, cmd2, ..., cmdn}, com $|C_k|$ = n, e que R_k = {rsp1, rsp2, ..., rspm}, com $|R_k|$ = m, a estrutura detalhada da mensagem do CLP_k para o CLP coordenador é como o apresentado a seguir:

- campo 1: seqüenciador de mensagens;

- campo 2: verificador de integridade;

- campo 3: modo de operação do CLP executor;

- campo 4: sinalização de ocorrência de erro na instância do FB RcvCmd implementada no CLP_k.

- campo 5: notificação do processamento do comando cmd1;

- campo 6: notificação do processamento do comando cmd2;

...

- campo w: notificação do processamento do comando cmdn (w = 4+n);

- campo x: número de ocorrências do evento não-controlável associado a rsp1 (x = 5+n);

- campo y: número de ocorrências do evento não-controlável associado a rsp2 (y = 6+n);

...

- campo z: número de ocorrências do evento não-controlável associado a rspm (z = 4+n+m).

Para ilustrar a inter-relação desejada entre CLPs, considere o caso elementar em que haja um único procedimento operacional implementado no CLP_k, O_k = {oα}. Este procedimento operacional está associado ao evento controlável α e, portanto, ao comando cmdα. A execução deste procedimento resulta na ocorrência do evento não-controlável β, o qual está associado à resposta rspβ. Assim, C_k = {cmdα} e R_k = {rspβ}. Deverá haver uma variável Booleana designada cmdα e uma variável do tipo *Unsigned Integer* designada rspβ, tanto no CLP coordenador quanto no CLP_k. Considere ainda que a variável gevt é a variável que sinaliza que foi realizado o tratamento de algum evento do subsistema cujo alfabeto é {α, β}.

Quando ocorre a geração do evento controlável α no CLP coordenador, ocorre também a ativação das variáveis cmdα e gevt neste CLP. Sempre que ocorre a ativação da variável cmdα, deve ser enviada uma mensagem do CLP coordenador para o CLP_k sinalizando este fato. Quando esta mensagem é recebida no CLP_k, deve ocorrer a ativação da variável cmdα neste CLP. A

ativação desta variável determina o início das atividades do procedimento operacional oα. Assim que estas atividades têm início, ocorre a desativação da variável cmdα no CLP_k. Sempre que esta variável é desativada, deve ser enviada uma mensagem do CLP_k para o CLP coordenador notificando o devido processamento do comando cmdα. Quando esta mensagem é recebida no CLP coordenador a variável cmdα deve ser desativada neste CLP. A desativação da variável cmdα confirma que o evento controlável α foi completamente tratado, resultando na desativação da variável gevt.

A execução das atividades pelo procedimento operacional oα irá resultar na ocorrência do evento não-controlável β. A cada ocorrência deste evento, a variável rspβ no CLP_k será incrementada em uma unidade. Sempre que é detectado que o valor associado à variável rspβ no CLP_k é maior do que zero, deve ser enviada uma mensagem deste CLP para o CLP coordenador informando o número de ocorrências do evento β pendentes para tratamento, ou seja, o valor associado à variável rspβ no CLP_k. Assim que a mensagem é enviada, à variável rspβ no CLP_k deve ser atribuído o valor zero. Quando esta mensagem é recebida no CLP coordenador, à variável rspβ neste CLP deve ser adcionado o número de ocorrências do evento β transmitido na mensagem.

A inter-relação entre CLPs, ou seja, a identificação de que há dados a serem transmitidos, bem como o tratamento destes dados através da manipulação das variáveis em C_k e R_k é estabelecida pelos FBs SendCmd, RcvCmd, SendRspCmdNtf e RcvRspCmdNtf.

Considerando o caso elementar introduzido anteriormente, a Figura 7.3 ilustra como estes FBs determinam o comportamento das variávies cmdα e rspβ no CLP coordenador e no CLP_k, bem como a troca de mensagens entre eles ao longo do tempo.

Inicialmente os dois CLPs estão trocando mensagens. As mensagens m1 e m3 são enviadas do CLP coordenador para o CLP_k, e as mensagens m2 e m4 no sentido oposto. A cada intervalo de tempo dado por (TimeOutIdle0 + T0) é enviada uma mensagem para sinalizar a vivacidade do CLP coordenador. No sentido oposto as mensagens são enviadas a intervalos (TimeOutIdlek + Tk). Nestas expressões T0 e Tk correspondem ao tempo transcorrido entre a ativação, nos FBs SendCmd e SendRspCmdNtf, do passo SendMessage e a ativação do passo Idle (ver Figura 6.3). Este valor incorpora o tempo para criar e transmitir a mensagem, receber a notificação de que a mensagem foi enviada com sucesso, bem como para ativar e desativar o passo MessageSent. Verifica-se da Figura 7.3 que (TimeOutIdle0 + T0) pode ser diferente de (TimeOutIdlek + Tk). Além disto os valores de T0 e Tk não são necessariamente constantes.

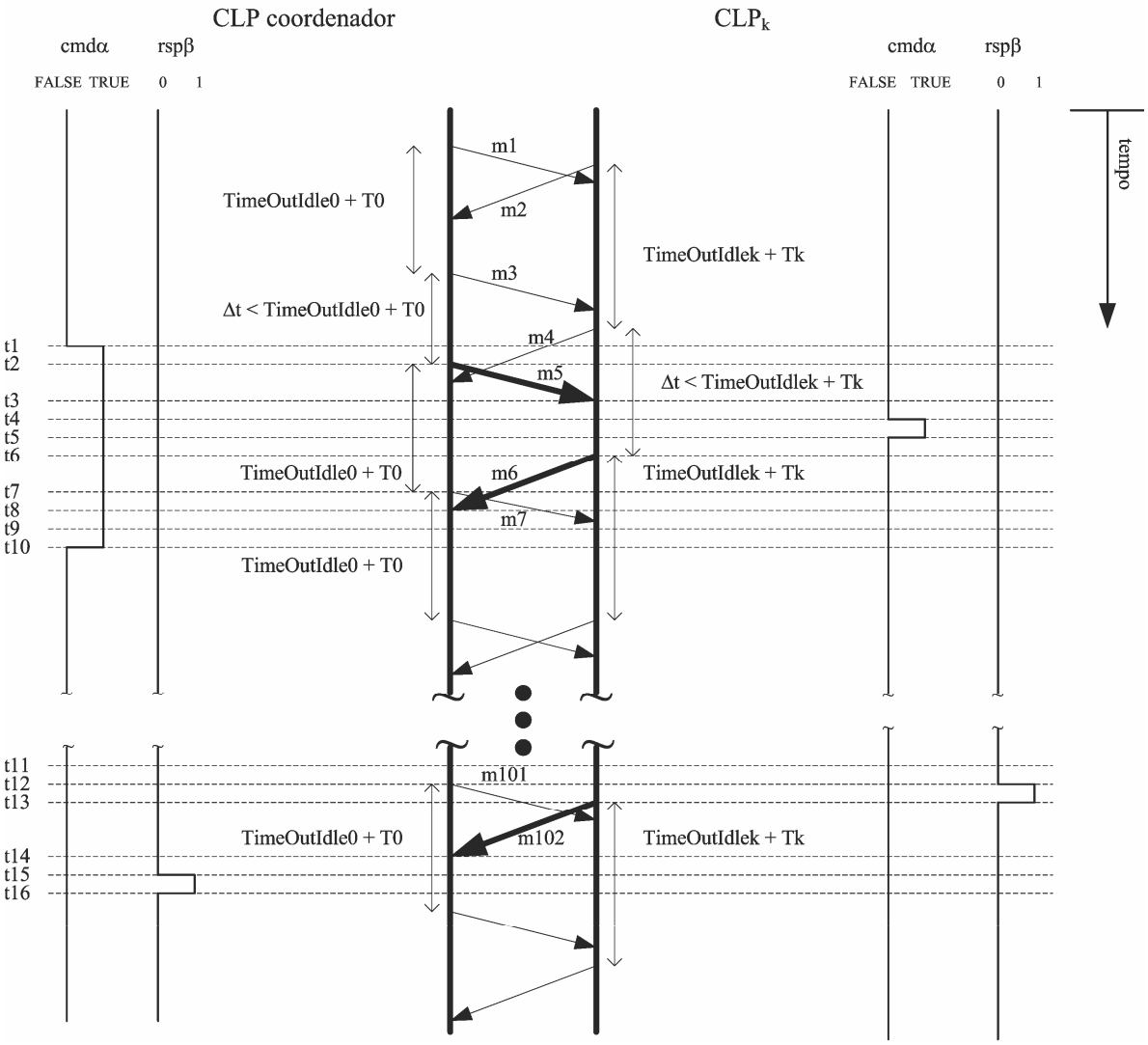


Figura 7.3 – Inter-relação e comunicação entre CLPs

No instante t1 ocorre, no nível Sistema Produto implementado no CLP coordenador, a geração do evento controlável α e a ativação do comando cmdα. No instante t2, após o FB SendCmd identificar a ativação de cmdα, a mensagem (m5) que sinaliza a ativação deste comando é criada e enviada do CLP coordenador para o CLP_k pelo FB SendCmd. Esta mensagem é recebida pelo FB RcvCmd no CLP_k no instante t3. A variável cmdα no CLP_k é ativada quando este FB valida e extrai os dados da mensagem (instante t4). No instante t5, ocorre o início da execução das atividades pelo procedimento operacional oα e a desativação de cmdα no CLP_k. No instante t6, após o FB SendRspCmdNtf identificar que a variável cmdα foi desativada, a mensagem (m6) que notifica o devido processamento deste comando é criada e enviada pelo referido FB. No instante t7 uma nova mensagem (m7) é enviada para sinalizar a vivacidade do CLP coordenador. No instante t8 o FB RcvRspCmdNtf recebe a mensagem m6. No instante t9, após validar e extrair o dados, o FB RcvRspCmdNtf notifica o FB SendCmd que o cmdα já foi devidamente processado pelo

procedimento operacional oα. No instante t10, após o FB SendCmd receber a notificação do processamento do comando cmdα, este FB desativa esta variável.

A partir deste instante de tempo os CLPs continuam trocando mensagens para sinalizar a vivacidade na rede. Enquanto isto o CLP_k executa as atividades previstas no procedimento operacional oα. A execução destas atividades resulta na ocorrência do evento não-controlável β, o que ocorre no instante t11. No instante t12 este procedimento operacional reconhece a ocorrência deste evento e incrementa em uma unidade a variável rspβ. No instante t13, após o FB SendRspCmdNtf reconhecer que o valor armazenado nesta variável é maior do que zero, é criada e enviada uma mensagem (m102) que informa o número de ocorrências do evento β desde o envio da última mensagem. Neste instante, O FB SendRspCmdNtf estabelece o valor zero à variável rspβ. No instante t14 esta mensagem é recebida pelo FB RcvRspCmdNtf. No instante t15, após validar e extrair os dados, este FB incrementa a variável rspβ com o valor recebido na mensagem. Finalmente, no instante t16 a ocorrência do evento β é sinalizada pelo nível Sistema Produto ao nível Supervisores Modulares e a variável rspβ é decrementada em uma unidade.

A análise desta figura conduz à conclusão, incorreta, de que o desempenho do sistema sob implementação distribuída é deteriorada em relação à implementação concentrada. Caso a implementação fosse realizada de forma concentrada, o início das atividades pelo procedimento operacional ocorreria logo após o instante t1, enquanto que sob implementação distribuída isto só ocorre no instante t5. De forma similar, sob implementação concentrada o tratamento da ocorrência do evento não-controlável ocorreria logo após o instante t12, enquanto que sob implementação distribuída isto só ocorre no instante t16. Porém, considerando o desempenho dos sistemas de comunicação atualmente disponíveis nos CLPs, tem-se que o tempo de resposta de um único atuador pode ser centenas ou milhares de vezes maior do que o atraso observado na referida figura, o que o torna insignificante. Além disto, sabendo que o tempo necessário para realizar cada ciclo de atualização em um CLP é proporcional ao número de operações realizadas ao longo deste ciclo e que, com a distribuição do controle parte das operações deixam de ser realizadas no CLP coordenador e passam a ser executadas no CLP executor, então a distribuição do controle resulta na redução do tempo de ciclo do CLP coordenador, o que resulta em um aumento da performance do sistema.

7.4 Os *Function Blocks* de comunicação

Os FBs SendCmd e SendRspCmdNtf são especializações do FB Sender introduzido no Capítulo 6. Os FBs RcvCmd e RcvRspCmdNtf são especializações do FB Receiver. A seção de código destes FBs são os SFCs Sender e Receiver apresentados nas Figuras 6.3 e 6.4. A

especialização destes FBs é realizada adicionando campos à estrutura da mensagem e detalhando os itens executados em cada uma das ações associadas aos passos destes SFCs. Também são adicionados novos itens a tais ações. Nas Figuras 7.4 e 7.5 são apresentadas estas ações. Nestas figuras os itens destacados com os números (1) a (4) são específicos, respectivamente, para o FB SendCmd, o FB SendRspCmdNtf, o FB RcvCmd e o FB RcvRspCmdNtf.

Conforme mencionado na Seção 6.3, não foram dedicados maiores esforços na busca ou desenvolvimento de algoritmos que realizem a verificação da integridade das mensagens. Desta forma, o item AddDataToMessage na ação action_SM dos FBs SendCmd e SendRspCmdNtf sempre atribuiu um valor arbitrário ao campo das mensagens referente ao verificador de integridade. Além disto, o item CheckIntegrity na ação action_VED dos FBs RcvCmd e RcvRspCmdNtf sempre atribuiu o valor lógico Verdadeiro à variável IntegrousMessage, o que significa que o conteúdo da mensagem está íntegro.

As próximas seções detalham os itens das ações dos FBs SendCmd, SendRspCmdNtf, RcvCmd e RcvRspCmdNtf. Também é detalhada a seção de declaração de variáveis destes FBs.

action_Init

```
SendError := FALSE;
Status := 0;
ResetMessage;
SetAllSendingCommandsToFalse; (1)
SetAllIntermediaryNotificationsToFalse; (2)
EmgSent := FALSE; (1) (2)
```

action_SIM

```
IF SendIntializerMessage.T = 0 THEN
    MsgSeq := 0;
    Message.msg1 := MsgSeq;
    SendToY(REQ := TRUE,
            ID := ID,
            R_ID := RID,
            SD_1 := Message);
ELSE
    SendToY(REQ := FALSE,
            ID := ID,
            R_ID := RID,
            SD_1 := Message);
    MessageSuccesfullySent := SendToY.DONE;
    CommunicationError := SendToY.ERROR;
    Status := SendToY.Status;
END_IF;
```

action_Idle

```
IF Idle.T = 0 THEN
    MsgSeq := MsgSeq + 1;
END_IF;
GetCmdNotification; (1)
MakeCmdNotification; (2)
```

action_SM

```
IF SendMessage.T = 0 THEN
    AddDataToMessage; (1), (2)
    SendToY(REQ := TRUE,
            ID := ID,
            R_ID := RID,
            SD_1 := Message);
ELSE
    SendToY(REQ := FALSE,
            ID := ID,
            R_ID := RID,
            SD_1 := Message);
    MessageSuccesfullySent := SendToY.DONE;
    CommunicationError := SendToY.ERROR;
    Status := SendToY.Status;
END_IF;
GetCmdNotification; (1)
MakeCmdNotification; (2)
```

action_MS

```
GetCmdNotification; (1)
MakeCmdNotification; (2)
IF MsgSeq >= MsgSeqMax THEN
    MsgSeq := 0;
END_IF;
```

action_TERD

```
SendError := TRUE;
```

- (1) aplicável apenas ao FB SendCmd
- (2) aplicável apenas ao FB SendRspCmdNtf
- SendToY é uma instância do Communication Function Block USEND

Figura 7.4 – Ações associadas aos passos do SFC Sender para especialização do FB Sender nos FBs SendCmd e SendRspCmdNtf

action_Init

```
RcvError := FALSE;
Status := 0;
OperMode := none; (3), (4)
RcvCmdError := FALSE; (4)
SetAllNotificationsToFalse; (4)
ResetMessage;
```

action_GFM

```
RcvFromX(EN_R:= TRUE,
         ID := ID,
         R_ID := RID);
MessageSuccesfullyReceived := RcvFromX.NDR;
CommunicationError := RcvFromX.ERROR;
Message := RcvFromX.RD_1;
Status := RcvFromX.Status;
```

action_FMR

```
CheckIntegrity;
IF IntegrousMessage THEN
    (* Integrous Message *)
    ExtractDataFromMessage;
    MsgSeq := MsgSeq + 1;
ELSE
    (* Corrupted Message *)
     RcvError := TRUE;
    STATUS := 2;
END_IF;

IF MsgSeq > MsgSeqMax THEN
    MsgSeq := 1;
END_IF;
```

action_GNM

```
IF GetNewMessage.T = 0 THEN (4)
    SetAllNotificationsToFalse;
END_IF;
RcvFromX(EN_R:= TRUE,
         ID := ID,
         R_ID := RID);
MessageSuccesfullyReceived := RcvFromX.NDR;
CommunicationError := RcvFromX.ERROR;
Message := RcvFromX.RD_1;
Status := RcvFromX.Status;
```

action_VED

```
(* High Priority Message *)
IF Message.msg3 = Emg THEN (3), (4)
    (* Emergency Mode *)
    OperMode := Emg;
    RETURN;
END_IF;
IF Message.msg1 = 0 THEN
   (* Initializer Message *)
    MsgSeq := 1;
    RcvError := FALSE;
    Status := 0;
    OperMode := Message.msg3;
    SetAllCommandsToFalse; (3)
    SetAllNotificationsToFalse; (4)
    RETURN;
END_IF;
(* Low Priority Messages *)
IF NOT RcvError THEN
    (* No errors up to now *)
    IF Message.msg1 = MsgSeq THEN
        (* Message Order Correct *)
        CheckIntegrity;
        IF IntegrousMessage THEN
            (* Integrous Message *)
            ExtractDataFromMessage;
            MsgSeq := MsgSeq + 1;
        ELSE
            (* Corrupted Message *)
            RcvError := TRUE;
            STATUS := 2;
       END_IF;
    ELSIF Message.msg1 < MsgSeq THEN
        (* Duplicated Message *)
    ELSIF Message.msg1 > MsgSeq THEN
        (* Message Lost *)
        RcvError := TRUE;
        STATUS := 3;
    END_IF;
END_IF;

IF MsgSeq > MsgSeqMax THEN
    MsgSeq := 1;
END_IF;
```

action_RESD

```
RcvError := TRUE;
IF NOT CommunicationError THEN
    STATUS := 1;
END_IF;
```

- (3) aplicável apenas ao FB RcvCmd
- (4) aplicável apenas ao FB RcvRspCmdNtf
- RcvFromX é uma instância do Communication Function Block URCV

Figura 7.5 – Ações associadas aos passos do SFC Receiver para especialização do FB Receiver nos FBs RcvCmd e RcvRspCmdNtf

7.4.1 FB SendCmd

Supondo que $|C_k| = n$, a Figura 7.6 detalha a seção de declaração de variáveis do FB SendCmd.

Na seção VAR_INPUT, a variável OperMode recebe o modo de operação do CLP coordenador. Conforme descrito na seção anterior, o FB RcvRspCmdNtf notifica o FB SendCmd quando cada um dos comandos em C_k foi processado pelo procedimento operacional correspondente no CLP_k. As variáveis em {cmdntf1, cmdntf2, ..., cmdntfn} recebem esta notificação. Para $i \in \{1, ..., n\}$ a variável $cmdntf_i$ recebe a notificação de que o comando cmd_i foi devidamente processado.

Na seção VAR, as variáveis em {sdgcmd1, sdgcmd2, ..., sdgcmdn} são utilizadas para memorizar se a ativação de cada um dos comandos em C_k já foi sinalizada ou não ao CLP_k. Para $i \in \{1, ..., n\}$ a variável $sdgcmd_i$ memoriza que foi sinalizada a ativação do comando cmd_i. A Figura 7.7 apresenta como é determinado se há novos dados a serem transmitidos do CLP coordenador para o CLP_k. De acordo com esta figura, caso haja algum comando ativo em C_k cuja ativação ainda não foi sinalizada ao CLP_k (por exemplo cmd1 AND NOT sdgcmd1), então há novos dados a serem transmitidos. Também há novos dados a serem transmitidos sempre que o modo de operação do CLP coordenador passa a ser o modo *Emergency* ((OperMode = Emg) AND NOT EmgSent). A variável EmgSent memoriza se esta informação já foi sinalizada ao CLP_k.

Na Figura 7.8 é detalhado o item ResetMessage constante da ação action_Init. Com a execução deste item cada campo da mensagem recebe um valor inicial pré-estabelecido. O item SetAllSendingCommandsToFalse é detalhado na Figura 7.9.

Sempre que o FB SendCmd recebe a notificação de que um determinado comando em C_k já foi processado pelo procedimento operacional correspondente, o comando em questão deve ser desativado. Também deve ser desativada a variável que memoriza se a sinalização da ativação deste comando já foi enviada ao CLP_K. O item GetCmdNotification (Figura 7.10) realiza estas ações. A adição de dados à mensagem é realizada pelo item AddDataToMessage, o qual é detalhado na Figura 7.11.

```
FUNCTION_BLOCK SendCmd                                  (* uma especialização do FB Sender *)

(* Seção de declaração de variáveis *)

TYPE                                                    (* Definição de tipos de variáveis *)

  MODE : (init, SI, PSI, PSIted, Man,Sup, Emg, none);   (* Modos de operação do CLP coord. *)

  MSG :                                                 (* mensagem coord. -> exec. *)
    STRUCT
      msg1 : INT;                                       (* sequenciador de mensagens *)
      msg2 : INT;                                       (* verificador de integridade *)
      msg3 : MODE;                                      (* modo de operação do CLP coord.  *)
      msg4 : BOOL;                                      (* sinalização de ativação do comando 1 * )
      msg5 : BOOL;                                      (* sinalização de ativação do comando 2 * )
      ...
      msgw : BOOL;                                      (* sinalização de ativação do comando n * )
    END_STRUCT;
END_TYPE

VAR_INPUT
  ID: COMM_CHANNEL;                                     (* ident. do canal de comunic.*)
  R_ID : STRING;                                        (* ident. do FB de comunic. parceiro*)
  OperMode : MODE;                                      (* modo de operação do CLP coord.  *)
  cmdntf1, cmdntf2, ..., cmdntfn : BOOL;                (* notificação de comandos processados *)
END_VAR

VAR_IN_OUT
  cmd1, cmd2, ..., cmdn : BOOL;                         (* comandos em Ck *)
END_VAR

VAR_OUT
  SendError: BOOL;                                      (* erro no envio de mensagem *)
  Status: INT;                                          (* especificação do erro *)
END_VAR

VAR
  Message : MSG;                                        (* mensagem *)
  MsgSeq : INT;                                         (* sequenciador local de mensagens *)
  MsgSeqMax : INT;                                      (* valor máx. do sequenciador de mensagens *)
  TimeOutIdle, TimeOutSend : Time;
  MessageSuccessufullySent : BOOL;                      (* mensagem enviada com sucesso *)
  CommunicationError : BOOL;                            (* erro no envio da mensagem *)
  NewDataToSend : BOOL;                                 (* novos dados a serem enviados *)
  sdgcmd1, sdgcmd2, ..., sdgcmdn : BOOL;                (* enviando sinal de ativ. dos comandos 1 a n *)
  EmgSent : BOOL;                                       (* notificação do modo Emergência realizada *)
END_VAR

(* Fim da seção de declaração de variáveis *)
```

Figura 7.6 – Seção de declaração de variáveis do FB SendCmd

```
(* DefineNDS)

NewDataToSend := (cmd1 AND NOT sdgcmd1) OR
                 (cmd2 AND NOT sdgcmd2) OR ...
                 ... OR (cmdn AND NOT sdgcmdn) OR
                        ((OperMode = Emg) AND NOT EmgSent)
```

Figura 7.7 – Determinação da existência de novos dados a serem transmitidos (DefineNDS)

```
(* ResetMessage *)

Message.msg1 := 0;
Message.msg2 := 0;
Message.msg3 := none;
Message.msg4 := FALSE;
Message.msg5 := FALSE;
...
Message.msgw := FALSE;
```

Figura 7.8 – Detalhamento do item ResetMessage

```
(* SetAllSendingCommandsToFalse  *)

sdgcmd1 := FALSE;
sdgcmd2 := FALSE;
...
sdgcmdn := FALSE;
```

Figura 7.9 – Detalhamento do item SetAllSendingCommandsToFalse

```
(* GetCmdNotification  *)

IF cmdntf1 THEN
   cmd1 := FALSE;          sdgcmd1 := FALSE;
END_IF;
IF cmdntf2 THEN
   cmd2 := FALSE;          sdgcmd2 := FALSE;
END_IF;
...
IF cmdntfn THEN
   cmdn := FALSE;          sdgcmdn := FALSE;
END_IF;
```

Figura 7.10 – Recebimento da notificação do processamento de comandos (GetCmdNotification)

```
(* AddDataToMessage *)

Message.msg1 := MsgSeq;
Message.msg2 := 0;
Message.msg3 := OperMode;
EmgSent := OperMode;
IF (cmd1 AND NOT sdgcmd1 ) THEN
   Message.msg4 := TRUE;          sdgcmd1 := TRUE;
ELSE
   Message.msg4 := FALSE;
END_IF;
IF (cmd2 AND NOT sdgcmd2 ) THEN
   Message.msg5 := TRUE;          sdgcmd2 := TRUE;
ELSE
   Message.msg5 := FALSE;
END_IF;
...
IF (cmdn AND NOT sdgcmdn ) THEN
   Message.msgw := TRUE ;         sdgcmdn := TRUE;
ELSE
   Message.msgw := FALSE;
END_IF;
```

Figura 7.11 – Adição de dados à mensagem (AddDataToMessage)

7.4.2 FB RcvCmd

A Figura 7.12 detalha a seção de declaração de variáveis do FB RcvCmd. Nesta figura é assumido que $|C_k| = n$.

A extração de dados das mensagens recebidas pelo CLP_k é realizada conforme apresentado na Figura 7.13. O item ResetMessage relativo ao FB RcvCmd é o apresentado na Figura 7.8.

```
FUNCTION_BLOCK RcvCmd                    (* uma especialização do FB Receiver *)

(* Seção de declaração de variáveis *)

TYPE                                                (* Definição de tipos de variáveis *)

  MODE : (init, SI, PSI, PSIted, Man,Sup, Emg, none);   (* Modos de operação do CLP coord. *)

  MSG :                                             (* mensagem coord. -> exec. *)
    STRUCT
      msg1 : INT;                                   (* sequenciador de mensagens *)
      msg2 : INT;                                   (* verificador de integridade *)
      msg3 : MODE;                                  (* modo de operação do CLP coord.  *)
      msg4 : BOOL;                                  (* sinalização de ativação do comando 1 * )
      msg5 : BOOL;                                  (* sinalização de ativação do comando 2 * )
      ...
      msgw : BOOL;                                  (* sinalização de ativação do comando n * )
    END_STRUCT;
END_TYPE

VAR_INPUT
  ID: COMM_CHANNEL;                                 (* ident. do canal de comunic.*)
  R_ID : STRING;                                    (* ident. do FB de comunic. parceiro*)
END_VAR

VAR_IN_OUT
  cmd1, cmd2, ..., cmdn : BOOL;                     (* comandos em Ck *)
END_VAR

VAR_OUT
  RcvError: BOOL;                                   (* erro na recepção de mensagens *)
  Status: INT;                                      (* especificação do erro *)
  OperMode : MODE;                                  (* modo de operação do CLP coordenador  *)
END_VAR

VAR
  Message : MSG;                                    (* mensagem *)
  MsgSeq : INT;                                     (* sequenciador local de mensagens *)
  MsgSeqMax : INT;                                  (* valor máx. do sequenciador de mensagens *)
  TimeOutGet : Time;
  MessageSuccessufullyReceived : BOOL;              (* mensagem recebida com sucesso *)
  CommunicationError : BOOL;                        (* erro na recepção da mensagem *)
  IntegrousMessage : BOOL;                          (* mensagem íntegra *)
END_VAR

(* Fim da seção de declaração de variáveis *)
```

Figura 7.12 – Seção de declaração de variáveis do FB RcvCmd

```
(* ExtractDataFromMessage *)

OperMode := Message.msg3;
cmd1 := cmd1 OR Message.msg4;
cmd2 := cmd2 OR Message.msg5;
...
cmdn := cmdn OR Message.msgw;
```

Figura 7.13 – Extração de dados das mensagens (ExtractDataFromMessagem)

7.4.3 FB SendRspCmdNtf

Supondo que $|C_k| = n$ e $|R_k| = m$, a Figura 7.14 detalha a seção de declaração de variáveis do FB SendRspCmdNtf.

As variáveis em {cmd1, cmd2, ..., cmdn} (na seção VAR_INPUT) e as variáveis em {cmd1prev, cmd2prev, ..., cmdnprev} e {ntf1, ntf2, ..., ntfn} (na seção VAR) são utilizadas para identificar o estado atual dos comandos em C_k, para memorizar o estado destas variáveis no ciclo de atualização anterior e estabelecer a notificação de que as variáveis em C_k passaram do nível lógico VERDADEIRO para o nível lógico FALSO, ou seja, de que o comando foi processado pelo procedimento operacional correspondente. Para $i \in \{1, ..., n\}$ cmd_i, $cmdprev_i$ e ntf_i estão inter-relacionadas.

Uma nova mensagem deve ser enviada do CLP_k para o CLP coordenador sempre que o valor associado a alguma variável em R_k é maior do que zero, ou que tenha ocorrido a desativação de qualquer comando em C_k desde o envio da última mensagem. Além disto, sempre que o modo de operação do CLP executor passa a ser o modo *Emergency*, é necessário enviar uma nova mensagem. A Figura 7.15 apresenta como é determinado se há novos dados a serem transmitidos.

Na Figura 7.16, o item MakeCmdNotification estabelece como as variáveis em {ntf1, ntf2, ..., ntfn} são ativadas. Na Figura 7.17 o item SetAllIntermediaryNotificationsToFalse desativa todas estas variáveis.

Na Figura 7.18 é detalhado o ítem ResetMessage constante da ação action_Init.

A adição de dados à mensagem é realizada pelo item AddDataToMessage, o qual é detalhado na Figura 7.19.

```
 FUNCTION_BLOCK SendRspCmdNtf          (* uma especialização do FB Sender *)

(* Seção de declaração de variáveis *)
TYPE                                   (* Definição de tipos de variáveis *)
   MODE : (init, SI, Run, Emg, none);  (* modos de operação do CLPk *)
   MSG :                               (* mensagem exec -> coord. *)
      STRUCT
         msg1 : INT;                   (* sequenciador de mensagens *)
         msg2 : INT;                   (* verificador de integridade *)
         msg3 : MODE;                  (* modo de operação do CLPk  *)
         msg4 : BOOL;                  (*ocorrência de erro na inst. do FB RcvCmd  *)
         msg5 : BOOL;                  (* notific. do processamento do comando 1 * )
         msg6 : BOOL;                  (* notific. do processamento do comando 2 * )
         ...
         msgw : BOOL;                  (* notific. do processamento do comando n * )
         msgx : INT;                   (* resposta 1 *)
         msgy : INT;                   (* resposta 2  *)
         ...
         msgz : INT;                   (* resposta m *)
      END_STRUCT;
END_TYPE

VAR_INPUT
   ID: COMM_CHANNEL;                   (* ident. do canal de comunic.*)
   R_ID : STRING;                      (* ident. do FB de comunic. parceiro*)
   OperMode : MODE;                    (* modo de operação do CLPk  *)
   RcvCmdError: BOOL;                  (*ocorrência de erro na inst. do FB RcvCmd  *)
   cmd1, cmd2, ..., cmdn : BOOL;       (* comandos em Ck *)
END_VAR

VAR_IN_OUT
   rsp1, rsp2, ..., rspm : INT;        (* respostas em Rk *)
END_VAR

VAR_OUT
   SendError: BOOL;                    (* erro no envio de mensagem *)
   Status: INT;                        (* especificação do erro *)
END_VAR

VAR
   Message : MSG;                      (* mensagem *)
   MsgSeq : INT;                       (* sequenciador local de mensagens *)
   MsgSeqMax : INT;                    (* valor máx. do sequenciador de mensagens *)
   TimeOutIdle, TimeOutSend : Time;
   MessageSuccessufullySent : BOOL;    (* mensagem enviada com sucesso *)
   CommunicationError : BOOL;          (* erro no envio da mensagem *)
   NewDataToSend : BOOL;               (* novos dados a serem enviados *)
   ntf1, ntf2, ..., ntfn : BOOL;       (* notificar o proc. dos comandos em Ck  *)
   cmd1prev, cmd2prev, ..., cmdnprev : BOOL;  (* estado lógico do comando no ciclo anterior *)
   EmgSent : BOOL;                     (* notificação do modo Emergência realizada *)
END_VAR

(* Fim da seção de declaração de variáveis *)
```

Figura 7.14 – Seção de declaração de variáveis do FB SendRspCmdNtf

```
(* DefineNDS *)

NewDataToSend := ( rsp1>0 OR rsp2>0 ... OR rspm>0 ) OR
                                    (ntf1 OR ntf2 OR ... OR ntfn ) OR
                                    ((OperMode = Emg) AND NOT EmgSent)
```

Figura 7.15 – Determinação da existência de dados a serem transmitidos (DefineNDS)

```
(* MakeCmdNotification  *)

IF ( cmd1prev AND NOT cdm1) THEN
   ntf1 := TRUE;
END_IF;
IF ( cmd2prev AND NOT cdm2) THEN
   ntf2 := TRUE;
END_IF;
...
IF ( cmdnprev AND NOT cdmn) THEN
   ntfn := TRUE;
END_IF;

cmd1prev := cmd1;
cmd2prev := cmd2;
...
cmdnprev := cmdn;
```

Figura 7.16 – Detalhamento do item MakeCmdNotification

```
(*SetAllIntermediaryNotificationsToFalse *)

ntf1:= FALSE;
ntf2 := FALSE;
...
ntfn := FALSE;
```

Figura 7.17 – Detalhamento do item SetAllIntermediaryNotificationsToFalse

```
(*ResetMessage  *)

Message.msg1 := 0;
Message.msg2 := 0;
Message.msg3 := none;
Message.msg4 := FALSE;
Message.msg5 := FALSE;
Message.msg6 := FALSE;
...
Message.msgw := FALSE;

Message.msgx := 0;
Message.msgy := 0;
...
Message.msgz := 0;
```

Figura 7.18 – Detalhamento do item ResetMessage

```
(* AddDataToMessage *)

Message.msg1 := MsgSeq;
Message.msg2 := 0;
Message.msg3 := OperMode;
EmgSent := OperMode;
Message.msg4 := RcvCmdError;
Message.msg5 := ntf1;                    ntf1 := FALSE;
Message.msg6 := ntf2;                    ntf2 := FALSE;
...
Message.msgw := ntfn;                    ntfn := FALSE;

Message.msgx := rsp1;                    rsp1 := 0;
Message.msgy := rsp2;                    rsp2 := 0;
...
Message.msgz := rspm;                    rspm := 0;
```

Figura 7.19 – Adição de dados à mensagem (AddDataToMessage)

7.4.4 FB RcvRspCmdNtf

A Figura 7.20 detalha a seção de declaração de variáveis do FB RcvRspCmdNtf. Nesta figura é assumido que $|C_k| = n$ e $|R_k| = m$.

A extração de dados das mensagens recebidas pelo CLP coordenador é realizada conforme apresentado na Figura 7.21. O item ResetMessage relativo ao FB RcvRspCmdNtf é o apresentado na Figura 7.18 e o item SetAllNotificationsToFalse é apresentado na Figura 7.22.

```
FUNCTION_BLOCK RcvRspCmdNtf              (* uma especialização do FB Receiver *)

(* Seção de declaração de variáveis *)
TYPE                                     (* Definição de tipos de variáveis *)
  MODE : (init, SI, Run, Emg, none);     (* modos de operação do CLPk *)
  MSG :                                  (* mensagem exec -> coord. *)
    STRUCT
      msg1 : INT;                        (* sequenciador de mensagens *)
      msg2 : INT;                        (* verificador de integridade *)
      msg3 : MODE;                       (* modo de operação do CLPk  *)
      msg4 : BOOL;                       (*ocorrência de erro na inst. do FB RcvCmd  *)
      msg5 : BOOL;                       (* notific. do processamento do comando 1 * )
      msg6 : BOOL;                       (* notific. do processamento do comando 2 * )
      ...
      msgw : BOOL;                       (* notific. do processamento do comando n * )
      msgx : INT;                        (* resposta 1 *)
      msgy : INT;                        (* resposta 2  *)
      ...
      msgz : INT;                        (* resposta m *)
    END_STRUCT;
END_TYPE

VAR_INPUT
  ID: COMM_CHANNEL;                      (* ident. do canal de comunic.*)
  R_ID : STRING;                         (* ident. do FB de comunic. parceiro*)
END_VAR

VAR_IN_OUT
  rsp1, rsp2, ..., rspm : INT;           (* respostas em Rk *)
END_VAR

VAR_OUT
  RcvError: BOOL;                        (* erro na recepção de mensagens *)
  Status: INT;                           (* especificação do erro *)
  OperMode : MODE;                       (* modo de operação do CLPk  *)
  RcvCmdError: BOOL;                     (*ocorrência de erro na inst. do FB RcvCmd  *)
  cmdntf1, cmdntf2, ..., cmdntfn : BOOL; (* notificação do proc. dos comandos em Ck *)
END_VAR

VAR
Message : MSG;                           (* mensagem *)
  MsgSeq : INT;                          (* sequenciador local de mensagens *)
  MsgSeqMax : INT;                       (* valor máx. do sequenciador de mensagens *)
  TimeOutGet : Time;
  MessageSuccessufullyReceived : BOOL;   (* mensagem recebida com sucesso *)
  CommunicationError : BOOL;             (* erro na recepção da mensagem *)
  IntegrousMessage : BOOL;               (* mensagem íntegra *)
END_VAR

(* Fim da seção de declaração de variáveis *)
```

Figura 7.20 – Seção de declaração de variáveis do FB RcvRspCmdNtf

```
(* ExtractDataFromMessage *)

OperMode:= Message.msg3;
RcvCmdError := Message.msg4;
cmdntf1 := Message.msg5;
cmdntf2 := Message.msg6;
...
cmdntfn := Message.msgw;

rsp1 := rsp1 + Message.msgx;
rsp2 := rsp2 + Message.msgy;
...
rspm := rspm + Message.msgz;
```

Figura 7.21 – Extração de dados das mensagens (ExtractDataFromMessagem)

```
(* SetAllNotificationsToFalse  *)

cmdntf1 := FALSE;
cmdntf2 := FALSE;
...
cmdntfn := FALSE;
```

Figura 7.22 – Detalhamento do item SetAllNotificationsToFalse

Exemplo 7.5)

Considerando a célula de manufatura introduzida na Seção 3.3 e a distribuição do controle proposta no Exemplo 7.1, a Figura 7.23 apresenta a ação action_Run do CLP executor CLP_2, no qual está implementado o conjunto de procedimentos operacionais O_2 = {oa2, iniG2}, sendo que C_2 = {cmda2, iniG2start} e R_2 = {rspb2, rspl2, rspm2, iniG2end};

A Figura 7.24 apresenta um segmento do código da ação action_Sup referente ao envio de mensagens do CLP_0 para o CLP_2.

```
(* action_Run *)

(* recepção de mensagens do CLP coordenador *)

     RcvCmdFrom0(ID := From0To2, R_ID = SendCmdTo2,
                                    cmd1: = cmda2, cmd2 := iniG2start);
     RcvErrorFrom0 := RcvCmdFrom0.RcvError;
     StatusFrom0 := RcvCmdFrom0.Status;
     CoordMode := RcvCmdFrom0.OperMode;

(* chamada para atualização dos procedimentos operacionais *)

     inst_oa2(Init := FALSE);
     inst_iniG2(Init := FALSE);

(* envio de mensagens ao CLP coordenador *)

     SendRspCmdNtfTo0(ID := From2To0, R_ID := RcvRspCmdNtfFrom2,
                                    OperMode := Exec2Mode, RcvCmdError:= RcvErrorFrom0,
                                    cmd1:= cmda2, cmd2 := iniG2start,
                                    rsp1 := rspb2, rsp2 := rspl2, rsp3 := rspm2, rsp4 := iniG2end);
     SendErrorTo0 := SendRspCmdNtfTo0.SendError;
     StatusTo0 : = SendRspCmdNtfTo0.Status;
```

Figura 7.23 – Ação action_Run do CLP executor CLP_2

```
(* segmento de código da ação action_Sup *)
...
(* recepção de mensagens do CLP executor 2 *)
     RcvRspCmdNtfFrom2(ID := From2To0, R_ID = SendRspCmdNtfTo0,
                                    rsp1 := rspb2, rsp2 := rspl2, rsp3 := rspm2, rsp4 := iniG2end);
     RcvErrorFrom2 := RcvRspCmdntfFrom2.RcvError;
     StatusFrom2 := RcvRspCmdntfFrom2.Status;
     Exec2Mode := RcvRspCmdntfFrom2.OperMode;
     RcvCmdErrorOn2 := RcvRspCmdntfFrom2.RcvCdmError;
     ntf1:= RcvRspCmdntfFrom2.cmdntf1;
     ntf2 := RcvRspCmdntfFrom2.cmdntf2;

(* envio de mensagens ao CLP executor 2 *)

     SendCmdTo2(ID := From0To2, R_ID := RcvCmdFrom0,
                                    OperMode := CoordMode,
                                    cmdntf1 := ntf1, cmdntf2:= ntf2
                                    cmd1 := cmda2, cmd2:= iniG2start);
     SendErrorTo2 := SendCmdTo2.SendError;
     StatusTo2 := SendCmdTo2.Status;
...
```

Figura 7.24 – Segmento da ação action_Sup do CLP coordenador CLP_0

♦ fim do exemplo 7.5 ♦

7.5 Considerações sobre o método

O método definido neste capítulo estende o método definido no Capítulo 5 no sentido que permite realizar a implementação distribuída em um conjunto de controladores. Com isto são alcançados os objetivos da distribuição apresentados no início deste capítulo:

i) Visto que o tempo necessário para executar um ciclo de atualização do CLP é função do número de operações avaliadas ao longo do ciclo, a distribuição do controle reduz o número de tais operações aumentando a performance do sistema. Este fato é particularmente significativo para os controladores onde estão implementados os procedimentos operacionais, pois é possível que num dado CLP seja implementado um número reduzido destes procedimentos, minimizando o tempo de processamento dos sinais oriundos dos sensores para atualização dos sinais a serem enviados aos atuadores;

ii) A implementação dos procedimentos operacionais em CLPs distintos daquele em que estão implementados os níveis Supervisores Modulares e Sistema Produto da Arquitetura de Controle proposta por Queiroz e Cury realiza a distribuição espacial do controle, permitindo que o controlador que efetivamente troca sinais com uma parcela do sistema a ser controlado esteja localizado próximo dos sensores e atuadores que constituem o sistema;

iii e *iv*) Em muitos casos, o sistema a ser controlado requer um controlador dotado de uma interface com um conjunto de características específicas. É possível que não exista um controlador que disponibilize uma interface que satisfaz simultaneamente todas estas características, porém, podem ser disponíveis diversos controladores que, se considerados em conjunto, satisfazem as características requeridas. A distribuição da implementação dos procedimentos operacionais permite a utilização de tais controladores. Outro aspecto a ser considerado é a capacidade de memória disponível nos controladores. A distribuição do controle permite a utilização de controladores de baixa capacidade de memória;

v) Há casos em que a estratégia de controle de um determinado equipamento já está implementada e validada em um determinado controlador, porém, é necessário coordenar a operação deste equipamento com outros equipamentos. Considerando que esta estratégia corresponde a procedimentos operacionais, sua reutilização no contexto abordado neste capítulo é bastante simplificada;

vi) A reconfiguração do sistema através da inclusão, remoção ou substituição de subsistemas é facilitada com o conceito de distribuição dos procedimentos operacionais;

vi) A distribuição da implementação dos procedimentos operacionais preserva a modularidade natural do sistema a ser controlado.

Considerando um sistema de grande porte e as características dos CLPs atualmente disponíveis no mercado, verifica-se que a capacidade de memória destes CLPs pode não ser compatível com a memória necessária para a implementação integral dos níveis Supervisores Modulares e Módulos do Sistema Produto da Arquitetura de Controle Supervisório de Queiroz e Cury. Além disto, é necessário integrar o sistema que realiza a execução da produção através da coordenação dos diversos subsistemas (função desempenhada pela referida arquitetura de controle) com sistemas de gerência e controle estatístico da produção, bancos de dados, sistemas de supervisão e interface com o usuário, dentre outros. Desta forma, é possível que o CLP coordenador seja realizado por um computador de uso geral no qual roda um SoftPLC ou outro sistema compatível. Tais computadores possuem capacidade de memória praticamente ilimitada em comparação aos CLPs atuais, além disto, a integração dos referidos sistemas empregando padrões como DDE (*dinamic data exchange*) ou OPC (*OLE for proccess control*) é bastante simplificada.

O método de implementação distribuída permite que o desenvolvimento do código seja realizado sem considerar previamente em qual controlador será realizada a implementação. Concluída esta etapa de desenvolvimento de código, a distribuição do controle pode ser definida com base nas características de interface e memória dos controladores efetivamente disponíveis para a implementação. Isto é válido desde que o código dos referidos controladores seja desenvolvido em um ambiente compatível com todos eles.

8. Conclusão e Perspectivas

Conforme apresentado em (COLLA *et al.*, 2006), (FREY e LITZ, 2000) e (TILBURY e KHARGONEKAR, 2000) a prática industrial corrente para desenvolvimento de sistemas de controle em sistemas de dinâmica dirigida a eventos discretos é, de forma geral, empírica com suporte na experiência adquirida pelo programador, e não adota métodos formais de modelagem do sistema nem tão pouco de síntese do controle.

A principal contribuição desta Tese de Doutorado é a apresentação de um método de implementação do controle de sistemas a eventos discretos com aplicação da Teoria de Controle Supervisório. Este método permite que a implementação do controle seja concentrada em um único controlador, possivelmente um controlador lógico programável (CLP), ou distribuída em um conjunto de CLPs. A Tese também apresenta um modelo de comunicação entre CLPs que garante a satisfação de um conjunto de propriedades consideradas relevantes à distribuição do controle. O método de implementação foi divulgado através de dois congressos (CBA 2006 - (VIEIRA *et al.*, 2006a) e ETFA 2006 - (VIEIRA *et al.*, 2006b)), também foi realizada submissão para publicação no periódico internacional *Control Engineering Practice* (VIEIRA *et al.*, 2007).

No Capítulo 5 desta Tese é apresentado um método de implementação da arquitetura de controle supervisório proposta por QUEIROZ e CURY (2002b) para o controle de sistemas a eventos discretos. Para adotar este método o comportamento do sistema e as especificações de controle são formalmente representados através de autômatos. A lei de controle, representada por um conjunto de supervisores, é obtida através da aplicação da Teoria de Controle Supervisório (RAMADGE e WONHAN, 1987), em particular uma extensão desta teoria, a Abordagem Modular Local (QUEIROZ, 2000) (QUEIROZ e CURY, 2000a, 2000b, 2002a).

O método de implementação é um procedimento sistemático que define o código a ser implementado no CLP. O código resultante da aplicação deste método é estruturado em diversos *Function Blocks* e um *Program*. Cada *Function Block* está diretamente associado a um autômato empregado na representação do comportamento do sistema ou a um supervisor. Isto preserva a modularidade natural do sistema, bem como das especificações de controle. Isto também facilita a interpretação, alteração e reutilização do código resultante.

O método de implementação é definido empregando linguagens de alto nível (*Sequential Function Chart* e *Structured Text*). Isto facilita sua apresentação e aplicação para quaisquer modelos empregados na representação do comportamento do sistema e supervisores obtidos com a aplicação da Teoria de Controle Supervisório. Contudo, o código pode ser implementado em qualquer linguagem de programação de CLP.

A aplicação do método de implementação resulta um código em conformidade com a norma internacional IEC 61131-3, a qual trata da programação de CLPs.

O método de implementação estabelece seis modos de operação distintos ao sistema. Estes modos de operação permitem a coordenação manual ou supervisionada do sistema a ser controlado, bem como a execução de procedimentos que conduzem o sistema a ser controlado para o estado inicial e a execução de procedimentos de emergência. Ao longo da apresentação do método de implementação são apresentadas indicações para o desenvolvimento de interfaces com o usuário

Em (FABIAN e HELLGREN, 1998), (BALEMI, 1992a), (DIETRICH *et al.*, 2002), (MALIK, 2002) e (PARK e CHO, 2006) são apresentados os principais problemas que podem se manifestar quando é realizada a implementação dos resultados obtidos com a aplicação da Teoria de Controle Supervisório. Nestas obras são definidas propriedades que, se satisfeitas pelo modelo que descreve o comportamento do sistema a ser controlado ou pela linguagem que descreve o comportamento ótimo sob supervisão, garantem que uma parcela de tais problemas não se manifesta. Ao final do Capítulo 5 é realizada uma discussão sobre a forma como o método de implementação garante que a totalidade de tais problemas não se manifeste.

As ferramentas computacionais atualmente disponíveis para realizar as operações associadas à Teoria de Controle Supervisório não incorporam a opção para verificação das propriedades mencionadas no parágrafo anterior. Considera-se relevante à difusão da aplicação industrial da Teoria de Controle Supervisório a incorporação a tais ferramentas da verificação das referidas propriedades. Ainda com este objetivo, é importante o desenvolvimento de ferramentas computacionais que realizem a geração automática do código do CLP com base nos modelos teóricos do comportamento livre dos diversos subsistemas e dos supervisores obtidos com a aplicação da Teoria de Controle Supervisório.

Em (SARDESAI *et al.*, 2006), (CENGIC *et al*., 2005) e (VYATKIN *et al.*, 2006) é apresentada a importância da distribuição do controle. Conforme COULOURIS *et al.* (2001), um sistema distribuído é aquele em que componentes em uma rede de computadores se comunicam e coordenam suas atividades apenas através da troca de mensagens.

No Capítulo 6 é apresentado um modelo de comunicação entre CLPs. A comunicação realizada utilizando este modelo satisfaz um conjunto de propriedades de interesse à distribuição do

controle. Dentre outras propriedades tem-se que a comunicação é confiável, isto é, a integridade do conteúdo das mensagens é preservada e não há perda nem duplicação de mensagens.

O referido modelo adota o serviço de comunicação *Programmed Data Acquisition* definido na norma internacional IEC 61131-5, a qual trata da comunicação entre CLPs. Contudo, caso este serviço não seja disponibilizado pelo sistema de comunicação utilizado, o modelo pode ser facilmente adaptado para o serviço de comunicação efetivamente disponível.

O modelo de comunicação é definido para uma configuração elementar, na qual um CLP envia mensagens e outro CLP as recebe. Contudo, este modelo é facilmente expandido para uma configuração com um número qualquer de CLPs, cada qual enviando e recebendo mensagens de quaisquer CLPs na rede.

No Capítulo 7 é realizada uma extensão do método de implementação definido no Capítulo 5. Isto tem por objetivo permitir que a implementação do controle seja distribuída em um conjunto de CLPs. Este método de implementação distribuída incorpora o modelo de comunicação definido no Capítulo 6.

Ao final do Capítulo 7 é apresentado como o método de implementação distribuída atinge os objetivos estabelecidos à distribuição do controle.

Nos sistemas industriais, o planejamento e execução da produção (definição do que, quando e quanto produzir) costumam ser realizados através de sistemas de execução da manufatura (*Manufacturing Execution Systems* - MES), possivelmente integrados a sistemas de planejamento de recursos (*Enterprise Resource Planning* - ERP). Conforme comentário recebido de um revisor de um congresso ao qual foi submetido um artigo "... o supervisor é como um guarda de trânsito que não dirige o sistema, mas que simplesmente evita que coisas indesejáveis aconteçam.... Na indústria, o controle é raro na restrição da ocorrência de fenômenos, mas sim, forçando que outros fenômenos aconteçam; é necessário um motorista que conduza o sistema ...". Este fato não é de todo válido. É possível incorporar especificações de controle que estabeleçam a execução da produção. Para satisfazer tais especificações é realizada a síntese e implementação de supervisores específicos. Porém, isto se torna mais complexo quanto maior for a diversidade da produção do sistema.

Deseja-se expandir a funcionalidade do método de implementação através da integração da arquitetura de controle supervisório de QUEIROZ e CURY com sistemas MES. Desta forma, a Teoria de Controle Supervisório é empregada para coordenar o sistema garantindo a satisfação de especificações de segurança e vivacidade, estabelecendo o que não pode ocorrer para que tais especificações não sejam violadas. Por outro lado, a execução da produção, ou seja, a definição de

ordens de produção é realizada pelo sistema MES. Tais ordens de produção devem ser submetidas à aprovação dos supervisores.

Esta integração pode ser realizada redefinindo elementos do método de implementação e incorporando uma interface com o sistema MES. O objetivo é que a definição de qual evento controlável a ser gerado a cada instante seja determinada com base nas ordens de produção estabelecidas pelo sistema MES e pela ação de controle dos supervisores. Neste caso o sistema MES é, então, um sistema externo ao controlador onde está implementada a arquitetura de controle supervisório. Outra possibilidade é alterar a arquitetura de controle supervisório (ver Figura 4.13) através da inclusão de um nível que realiza a função de um sistema MES. GASPER *et al.* (2005) propõem uma arquitetura de controle que remete a este conceito.

O método de implementação apresentado no Capítulo 7 permite a distribuição do nível Procedimentos Operacionais da arquitetura de controle de QUEIROZ e CURY. Em (VIEIRA, 2004) foi proposto um método de implementação que permite que a distribuição atinja, também, os níveis Supervisores Modulares e Sistema Produto da referida arquitetura de controle. No Capítulo 7 foram apresentados os objetivos da distribuição do controle. Recomenda-se que, à luz de tais objetivos e considerando as características dos sistemas industriais, seja avaliada a relevância da distribuição dos níveis Supervisores Modulares e Sistema Produto. Caso seja constatada a relevância desta distribuição recomenda-se que o método de distribuição apresentado no Capítulo 7 deste documento seja expandido, possivelmente empregando os conceitos introduzidos no método apresentado em (VIEIRA, 2004).

Conforme mencionado na Seção 4.1.5, verifica-se uma intensa atividade do meio acadêmico na investigação de aplicações da norma internacional IEC 61499 na estruturação de sistemas distribuídos. Porém conforme SUNDER *et al.* (2006) "*widespread adoption of this technology by industry is just in its infancy*.". Como continuidade desta pesquisa deseja-se investigar a aplicação do método de implementação definido nesta Tese a controladores compatíveis com a referida norma.

Referências Bibliográficas

ATTIÉ, S.S.; 1998. *Automação Hidráulica e Pneumática Empregando a Teoria de Sistemas a Eventos Discretos.* Florianópolis. Dissertação (Mestrado em Engenharia Mecânica) - Centro Tecnológico, Universidade Federal de Santa Catarina.

BALEMI, S.; 1992a. Control of Discrete Event Systems: Theory and Application. Thesis (Doctor of Technical Sciences) - Swiss Federal Institute of Technology. Zurich.

————————; 1992b. Communication delays in connections of input/output discrete event processes. *In: Proc. 31st. Conference on Decision and Control*. USA, p. 3374-3379.

BALEMI, S.; HOFFMANN, G.J.; GYUGYI, P.; WONG-TOI, H.; FRANKLIN, G.F.; 1993. Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control*, v. 38, n. 7, p. 1040-1059.

BAUER, N.; HUUCK, R.,LUKOSCHUS, B., ENGELL, S.; 2004. A unifying semantics for sequential function charts. *In: Integration of Software Specification Techniques for Applications in Engineering - Lecture notes in computer series*. Germany: Springer, p. 400-418.

BOLLMANN, A.; 1996. *Fundamentos da automação industrial pneutrônica. Brasil: ABHP.*

BONG-SUK, K.; KWANG-HIYUN, C. 2001. Design of PLCs for automated industrial systems based on discrete event models. *In: IEEE International Symposium on Industrial Electronics*. Pusan, Korea, v. 3, p. 1431-1434.

BRANDIN, B.A.; 1996. The real-time supervisory control of an experimental manufacturing cell. *IEEE Transactions on Robotics and Automation*, v. 12, n. 1, p.1 – 14.

BRANDIN, B.A.; CHARBONNIER, F.E.; 1994. The supervisory control of the automated manufacturing system of the AIP. *In: 4th International Conference on Computer Integrated Manufacturing and Automation Technology*, p. 319-324.

CARROLL, J.; LONG, D.; 1989. *Theory of Finite Automata*. USA: Prentice-Hall.

CASSANDRAS, C.G.; LAFORTUNE S.; 1999. *Introduction to Discrete Event Systems*. USA : Kluwer Academic Publishers.

CENGIC, G.; Akesson, K.; LENNARTSON, B.; CHENGYIN, Y.; FERREIRA, P.; 2005. Implementation of full synchronous composition using IEC 61499 function blocks. *In: IEEE International Conference on Automation Science and Engineering*.

CENGIC, G.; LJUNGKRANTZ, O.; AKESSON, K.; 2006. Formal modeling of Function Block applications running in IEC 61499 execution runtime. *In: IEEE International Conference on Emerging Technology and Factory Automation*. Prague, Czech Republic, p. 1269-1276.

CHARBONNIER, F.; ALLA, H.; DAVID, R.; 1999. The supervised control of Discrete-event dynamic systems**.** *IEEE Transactions on Control Systems Technology*, v. 7, n. 2, p. 175-187.

CHRISTENSEN, J.H.; 2000a. Basic concepts of IEC 61499. *In: Proc. Fachtagung Verteilt Automatisierung.* Magdeburg, Germany, p. 55-62

————————; 2000b. Design patterns for system enginering. *In: Proc. Fachtagung Verteilt Automatisierung.* Magdeburg, Germany, p. 63-71.

COLLA, M.; BRUSAFERRI, A.; CARPANZANO, E.; 2006. Applying the IEC 61499 Model to the shoe manufacturing sector. *In: IEEE International Conference on Emerging Technology and Factory Automation*. Prague, Czech Republic, p. 1301-1308.

CÔTÉ, D., ST-DENIS, R., KERJEAN, S.; 2005. Generative Programming for Programmable Logic Controllers. *In: 10th IEEE International Conference on Emerging Technologies and Factory Automation*. Catania, Italy, p. 741-748.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; 2001. *Distributed System: Concepts and design*. England: Pearson Education.

CURZEL, J.L.; LEAL, A.B.; 2006. Implementação de controle supervisório em linguagem Ladder para uma célula flexível de manufatura didática. *In: XVI Congresso Brasileiro de Automática*. Bahia, Brasil, p.2700-2705.

CURY, J.E.R; 2001. *Teoria de Controle Supervisório de Sistemas a Eventos Discretos*. Centro Tecnológico, Universidade Federal de Santa Catarina.

DE NEGRI, V.J.; 1996. *Estruturação da modelagem de sistemas automáticos e sua aplicação a um banco de testes para sistemas hidráulicos.* Florianópolis. Tese (Doutorado em Engenharia Mecânica) – Centro Tecnológico, Universidade Federal de Santa Catarina.

DIETRICH, P.; MALIK, R.; WONHAM, W.M.; BRANDIN, B.; 2002. Implementation considerations in supervisory control. *In: Synthesys and control of discrete event systems*. Kluwer Academic Publishers, p. 185-201.

FABIAN, M.; HELLGREN, A.; 1998. PLC-based implementation of supervisory control for discrete event systems. *In: 37th IEEE Conference on Decision and Control*, v. 3, p. 3305-3310.

FERRARINI, L.; VEBER, C.; 2004. Implementation approaches for the execution model of IEC 61499 applications. *In: Proc. 2nd IEEE International Conference on Industrial Informatics*, Berlin, Germany, p. 612-617.

FERREIRA, A.B.de H.; 1999. *Novo Aurélio – Dicionário da Língua Portuguesa – Século XXI*. 3ª ed. Brasil: Editora Nova Fronteira.

FREY, G.; LITZ, L. 2000. Formal methods in PLC programming. *In: IEEE Conference on systems Man and Cybernetics*, p. 2431-2436.

FREY, G.; HUSSAIN, T.; 2006. Modeling Techniques for distributed control systems based on the IEC 61499 Standard - Current approaches and open problems. *In: Proc. of the 8th. Workshop on Discrete Event Systems*. Ann Arbor, USA, p. 176-181.

GASPER, M., MATKO, D.; 2002. Discrete Event Control Theory Applied to PLC Programming. Automatika, vol. 43 n° 1-2, Croatie, p. 21-28.

GASPER, M. GRADISAR, D., MATKO, D.; 2005. IEC 61131-3 Compliant control code generation from discrete event models. *In: 13th. Mediterranean Conference on Control and Automation.* Limassol, Cyprus, p. 346-351.

GOLASZEWSKI, C. H., RAMADGE, P. J.; 1987. Control of discrete event processes with forced events. *In: IEEE Proceedings of the 26th Conference on Decision and Control*, pp. 247-252.

HASDENIR, T., KURTULAN, S., GÖREN, L.; 2004. Implementation of Local Modular Supervisory Control for a pneumatic system using PLC. *In: 7th International Workshop on Discrete Event Systems*, p. 25-30.

HELLGREN, A.; 2000. Modelling and Implementation Aspects of Supervisory Control. Sweden . Licentiate Thesis – Department of Signals and Systems, Chalmers University of Technology.

————————; 2002. *On the Implementation of Discrete Event Supervisory Control with focus on flexible manufacturing systems*. Sweden. PhD. Thesis – Department of Signals and Systems, Chalmers University of Technology.

HELLGREN, A.; LENNARTSON, B.; FABIAN, M.; 2002. Modelling and PLC-based implementation of modular supervisory control Discrete Event Systems. In: 6th

International Workshop in Discrete Event Systems. Zaragoza, Spain: Kluwer Academic Publishers, p. 371-376.

HELLGREN, A.; FABIAN, M.; LENNARTSON, B.; 2005. On the execution of sequential function charts. *Control Engineering Practice*, v. 13 p. 1283-1293.

HOHPE, G.; WOOLF, B.; 2004. *Enterprise Integration Patterns: Designing, Building and Deploying Messaging Solutions.* USA: Addison-Wesley.

HOARE, C.A.R ; 1985. *Communicating Sequential Processes.* Prentice Hall International.

HOPCROFT, J.E.; MOTWANI, R.; ULLMAN, J.D.; 2001. *Introduction to Automata Theory, Languages, and Computation.* 2nd. ed. USA: Addison-Wesley.

HUSSAIN, T.; FREY, G.; 2005. Migration of a PLC controller to an IEC 61499 compliant distributed control system: Hands-on experiences. *In: IEEE International Conference on Robotics and Automation*. Barcelona, Spain, p. 3995-4000.

IEC; 2000. *International Standard IEC 61131-5, Programmable Logic Controllers – Part 5: Communications.*

IEC; 2002. *International Standard IEC 60848, GRAFCET specification language for sequential function charts.*

IEC; 2003. *International Standard IEC 61131-3,* Programmable Logic Controllers – Part 3: Programming Languages.

IEEE. *Standard Dictionary of Electrical and Electronic Terms*.

ISAGRAF; 2007. http://isagraf.com. Acessado em agosto/2007.

JOHN, K.H.; TIEGELKAMP, M.; 2001. *IEC 61131-3: Programming Industrial Control Systems*. Germany: Springer-Verlag.

KUMAR, R.; GARG, V.; 1995. *Modeling and Control of Logical Discrete Event Systems.* USA: Kluwer Academic Publishers.

LASTRA, J.L.M.; LOBOV, A; 2005. An IEC 61499 application generator for scan-based industrial controlers. *In: 3rd International Conference on Industrial Informatics*, p. 80-85.

LAUZON, S.C.; MA, A.K.L.; MILLS, J.K.; BENHABIB, B.; 1996. Application of discrete-event-system theory to flexible manufacturing. *IEEE Control Systems Magazine*, v. 16, n. 1, p. 41-48.

LAUZON, S.C.; MILLS, J.K.; BENHABIB, B.; 1997. An implementation methodology for the supervisory control of flexible manufacturing workcells. *SME Journal of Manufacturing Systems*, v. 16, n. 1, p. 91-101.

LEDUC, R.J.; 1996. *PLC Implementation of a DES Supervisor for a Manufacturing Testbed: An Implementation Perspective*. Canada. Master Thesis – Dep. of Electrical and Computer Engineering, University of Toronto.

LEWIS, R.W.; 1998. *Programming industrial control systems using IEC 1131-3 - Revised Edition,* London, United Kingdon: The Institution of Electrical Engineers.

LEWIS, R.; 2001. *Modelling control systems using IEC 61499*, London, United Kingdon: The Institution of Electrical Engineers.

LIU, J.; DARABI, H.; 2002. Ladder logic implementation of Ramadge-Wonham supervisory controller. In: 6th International Workshop in Discrete Event Systems. Zaragoza, Spain: Kluwer Academic Publishers, p. 383-392.

MALIK, P.; 2002. Generating Controllers from Discret-Event Models. In: F. Cassez, C. Jard , F. Laroussinie, and M. D. Ryan, editors, MOVEP'2002, pp. 337-342.

MINHAS, R.S.; 2002. *Complexity Reduction in Discrete Event Systems.* Canada. PhD. Thesis – Dep. of Electrical and Computer Engineering, University of Toronto.

MUSIC, G.; MATKO, D.; 2005. Combined synthesis/verification approach to programmable logic control of a production line. *IFAC World Congress*. Prague, Czech Republic.

OHMAN, M.; JOHANSSON, S.; ARZEN, K.E.; 1998. Implementation aspects of the PLC standard IEC 61131-3, v. 6, n. 4, p. 547-555.

PARK, S.J.; CHO, K.H.; 2006. Delay-robust supervisory control of discrete-event systems with bounded communication delays. *IEEE Transactions on Automatic Control*, v. 51, n. 5, p. 911-915.

PETIG, M.; 2000. The way to distributed PLCs. *Dedicated Systems Magazine*, Q2, 2000.

PLCOPEN; 2007. http://www.plcopen.org. Acessado em agosto/2007.

QUEIROZ, M.H. de; 2000. *Controle Supervisório Modular de Sistemas de Grande Porte.* Florianópolis. Mestrado (Dissertação em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.

————————; 2004. *Controle Supervisório Modular e Multitarefa de Sistemas Compostos.* Florianópolis. Tese (Doutorado em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.

QUEIROZ, M.H. de; CURY, J.E.R.; 2000a. Modular supervisory control of large scale discrete-event systems. *In: Proceedings of the 5th International Workshop on Discrete Event Systems: Analysis and Control*. Ghent, Belgium: Kluwer Academic Publishers, p. 103-110.

————————; 2000b. Modular control of composed systems. *In: Proceedings of the American Control Conference*. Chicago, USA.

————————; 2002a. Controle supervisório modular de sistemas de manufatura. *Revista controle & automação*, v. 13, n. 2, p. 115-125.

————————; 2002b. Synthesis and implementation of local modular supervisory control for a manufacturing cell. In: 6th International Workshop in Discrete Event Systems. Zaragoza, Spain: Kluwer Academic Publishers, p. 377-382.

RAMADGE, P. J.; WONHAM, W. M.; 1987. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, v. 25, n. 1, p. 206–230.

————————; 1989. The control of discrete event systems, *Proc. of IEEE, Special Issue on Discrete Event Dynamic Systems*, v. 77, n. 1, p. 81-98.

RAMIREZ, A.; ZHU, S.C.; BENHABIB, B.; 1999. Moore Automata for flexible routing and flow control in manufacturing workcells. *In: International Symposium on Computational Intelligence in Robotics and Automation. Monterey, CA, p. 119-124.*

RAMIREZ-SERRANO, A.; ZHU, S.C.; CHAN, S.K.H.; CHAN, S.S.W.; BENHABIB, B.; 2000. A hybrid PC/PLC architecture for manufacturing system control-implementation. *In: IEEE International Conference on Systems, Man, and Cybernetics*, v. 3, p. 1697-1702.

SANTOS, E.A.P.; VIEIRA, A.D.; BUSETTI, M.A.; 2006. Controle de um sistema integrado de manufatura baseado na Teoria de Controle Supervisório. *In: Congresso Brasileiro de Automática,* Bahia, Brasil, p. 1181-1186.

SARDESAI, A.R.; MAZHARULLAH, O.; VYATKIN, V.; 2006. Reconfiguration of mechatronic systems enabled by IEC 61499 Function Blocks. *In: Proc. Australasian Conference on Robotics and Automation*. Auckland, New Zealand.

SIEMENS; 2007. http://support.automation.siemens.com. Acessado em agosto/2007.

SILVA, D.B.; SANTOS, E.A.P., VIEIRA, A.D..; PAULA, M.A.; 2007a. Aplicação da Teoria de Controle Supervisório em Sistemas Automatizados de Manufatura Multi Produtos. *In: Simpósio Brasileiro de Automação Inteligente.* Brasil.

————————; 2007b. Application of the supervisory contrl theory to automated systems of multi-product manufacturing. *In: 12th IEEE International Conference on Emerging Technologies and Factory Automation*. Greece.

STALLINGS, W.; 1999. *Data and computer communications*. USA : Prentice Hall, 6th. ed.

SU, R.; WONHAM, W.M.; 2004. Supervisor Reduction for Discrete-Event Systems. *Discrete Event Dynamic Systems*, v. 14 n. 1, p. 31-53.

SUNDER, C.K.; ZOITL, A.; CHRISTENSEN, J.H.; VYATKIN, V.; BRENNAN, R.; VALENTINI, A.; FERRARINI, L.; STRASSER, T.; LASTRA,J.L.M.; AUINGER, F.; 2006. Usability and Interoperability of IEC 61499 based distributed automations systems. *In: Proc. 4 th InternationalIEEE Conference on Industrial Informatics.* Singapore

TEIXEIRA, C.A.; LEAL, A.B.; SOUZA, A.H.; 2006. Implementação de supervisores em microcontroladores: Uma abordagem baseada na teoria de controle de sistemas a eventos discretos. *In: XVI Congresso Brasileiro de Automática*. Bahia, Brasil, p. 2772-2777.

TILBURY, D.; KHARGONEKAR, P.; 2000. *Challenges and Opportunities in Logic Control for Manufacturing Systems*. Report of the NSF Workshop held at the University of Michigan Ann Arbor.

THRAMBOULIDIS, K.; 2005. IEC 61499 in factory automation. *In*: *Proc. IEEE Int. Conf. on Industrial Electronics, Technology and Automation*. Bridgeport, USA.

TORRICO, C.R.C.; 1999. *Implementação de Controle Supervisório de Sistemas a Eventos Discretos Aplicado a Processos de Manufatura*. Florianópolis. Dissertação (Mestrado em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.

VAZ, F., WONHAM, W.M.; 1986. On supervisory reduction in discrete event systems. *International Journal of Control*, v. 44, n. 2, p. 475-491.

VIEIRA, A.D.; 2004. *Contribuições à implementação de sistemas de controle supervisório. Florianópolis*. Qualificação de doutorado – Centro Tecnológico, Universidade Federal de Santa Catarina.

VIEIRA, A.D; CURY, J.E.R.; QUEIROZ, M.H. de; 2004a. Distribuição de estrutura de controle supervisório de sistemas a eventos discretos. *In: XV Congresso Brasileiro de Automática.* Gramado – RS, Brasil.

————————; 2004b. Distributed Supervisory Control of Discrete Event Systems Based on Local Modular Control. *In: VI Conferência Internacional de Aplicações Industriais.* Joinville – SC, Brasil, p. 111-116.

————————; 2006a. Um modelo para implementação de controle supervisório em controladores lógico programáveis. *In: XVI Congresso Brasileiro de Automática*. Bahia, Brasil, p.1974-1979.

————————; 2006b. A Model for PLC Implementation of Supervisory Control of Discrete Event Systems. *In 11th IEEE International Conference on Emerging Technologies and Factory Automation*, p. 225 - 232.

————————; 2007. A model for PLC implementation of supervisory control of discrete event systems. *Control Engineering Practice*. Submetido para publicação.

VYATKIN, V.; 2006. *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design*. USA: Instrumentation Society of America.

VYATKIN, V.; HIRSCH, M.; HANISCH, H.M.; 2006. Systematic design and implementation of distributed controllers in industrial automation. *In: IEEE International Conference on Emerging Technology and Factory Automation*. Prague, Czech Republic, p. 633-640.

WILLNER, Y; HEYMANN, M.; 1991. Supervisory Control of Concurrent Discrete-event Systems. *International Journal of Control*, v. 54, n. 5, p. 1143-1169.

WILSON, B.; 1990. *Systems: concepts, methodologies and applications.* 2nd ed. UK: John Wiley & Sons.

WONHAM, W. M.; 2003. *Notes on Control of Discrete-Event Systems*. Canada. Dept. of Electrical and Computer Engineering, University of Toronto.

WONHAM, W. M; RAMADGE, P. J.; 1987. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, v. 25, n. 3, p. 637–659.